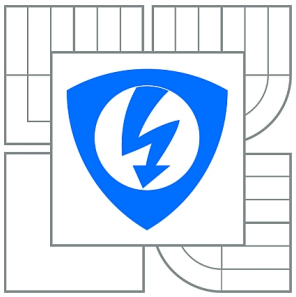
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH A REALIZACE AUTENTIZAČNÍ METODY PRO PŘÍSTUP K WEBOVÉ SLUŽBĚ V PROSTŘEDÍ .NET

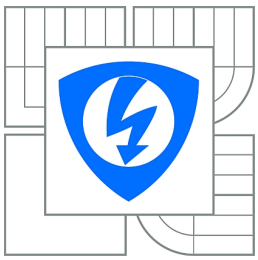
DESIGN AND IMPLEMENTATION OF AUTHENTICATION METHOD FOR ACCESSING WEB SERVICE
IN .NET ENVIRONMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. LUKÁŠ TENORA
AUTHOR

VEDOUCÍ PRÁCE — Ing. OTTO DOSTÁL, CSc.
SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Lukáš Tenora ***ID:*** 78419
Ročník: 2 ***Akademický rok:*** 2009/2010

NÁZEV TÉMATU:

Návrh a realizace autentizační metody pro přístup k webové službě v prostředí .NET

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte autentizační principy přístupu k webovým službám. Navržený princip implementujte do technologie session ASP.NET. Metodu zhodnoťte s ohledem k nasazení v počtu stovek a tisíců současných přístupů ke službě. Hodnotící parametry jednotlivých variant řešení jsou bezpečnost a klientská data zabezpečená silným algoritmem na serveru.
Vypracujte softwarový autentizační webový přístup v jazyce .NET s využitím session. Vypracujte přístupový portál na základě nejvhodnějšího řešení.

DOPORUČENÁ LITERATURA:

[1] Stallings, W.: Cryptography and Network Security: Principles and Practice. Prentice Hall, Englewood Cliffs 2003. ISBN: 0-13-091429-0
[2] ASP.NETa ADO.NET 2.0, CPress
[3] Vytváříme zabezpečené aplikace v Microsoft ASP.NET, CPress

Termín zadání: 29.1.2010 ***Termín odevzdání:*** 26.5.2010

Vedoucí práce: Ing. Otto Dostál, CSc.

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Práce se zabývá problematikou autentizačních metod pro přístup k webovým službám. V první části jsou vysvětleny autentizační metody a možná bezpečnostní rizika. V druhé části práce je popsán návrh a realizace vlastní autentizace k webové službě v prostředí .NET.

KLÍČOVÁ SLOVA

Bezpečnost, autentizace, ASP.NET.

ABSTRACT

Work deals with the authentication methods for access to Web services. The first part of the authentication methods are explained and possible safety risks. The second part describes the design and implementation of custom authentication Web service environment.. NET

KEYWORDS

Security, Authentication, ASP.NET.

TENORA L. *Návrh a realizace autentizační metody pro přístup k webové služběv prostředí .NET*. Brno: Vysoké učení technické. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2010. 56 s., 0 s. příloh. Diplomová práce. Vedoucí práce byl Ing. OttoDostál, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma Návrh a realizace autentizační metody pro přístup k webové službě v prostředí .NET jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 26.Května 2010

Lukáš Tenora
(podpis autora)

PODĚKOVÁNÍ

Děkuji konzultantudiplomové práce Ing. Stanislavu Uchytilovi, Ph.D. za velmi užitečnou metodickou pomoc a cenné rady při zpracování diplomové práce.

V Brně dne 26. Května 2010

Lukáš Tenora
(podpis autora)

SEZNAM ZKRATEK

AES	Advanced Encryption Standard (Pokročilý šifrovací standard)
CRL	Common Language Runtime (běhové prostředí pro aplikace pro .NET)
HTTP	Hypertext Transfer Protokol
IIS	Internet Information Service (Internetová informační služba)
JIT	Just in Time
LINQ	Language Integrated Query (Dotazovací jazyk)
MD5	Message-Digest algorithm 5 (Algoritmus pro zpracování zpráv č. 5)
MSIL	Microsoft Intermadiate Language (procesorově nezávislý jazyk)
NIST	Národní institut pro standardizaci a technologie
SHA	Secure Hash Algorithm (Bezpečný hašovací algoritmus)
SQL	Structured Query Language (Dotazovací jazyk)
WCF	Windows Communication Foundation
WPF	Windows Presentation Foundation
WWF	Windows Workflow Foundation

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Cílem práce bylo rozebrat problematiku autentizačních principů webových služeb. Nejprve bylo teoreticky rozebráno zabezpečení webových služeb, dále jsem se věnoval webovému serveru IIS a technologii .NET. Následně jsem rozebral autentizační principy a možné útoky na webové služby a provedl jejich zhodnocení.

Praktická část se věnuje vlastnímu návrhu autentizační metody a její realizaci v prostředí .NET. Po konzultaci s vedoucím práce byla zvolena metoda HTTP autentizace spojená s formulářovou autentizací vůči uživatelským účtům uloženým v SQL databázi.

1 ZABEZPEČENÍ WEBOVÝCH SLUŽEB

K zajištění bezpečnosti webových služeb se využívají kryptografické metody, které pro zabezpečení poskytují následující služby:

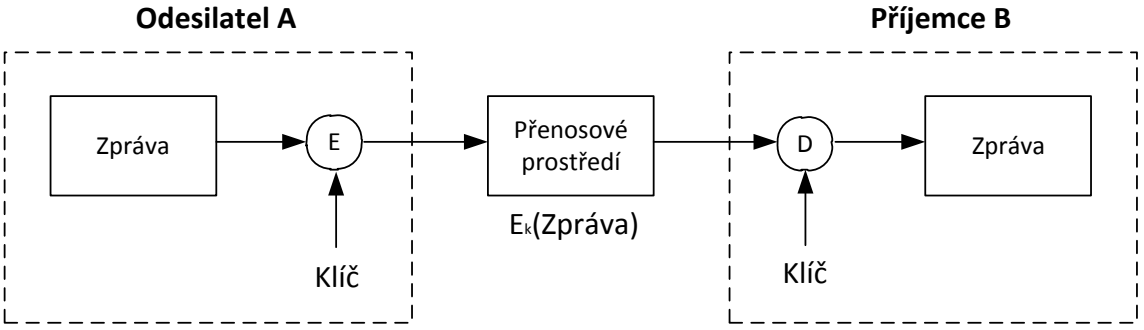
- **Důvěrnost**, zabezpečuje utajení informace před neoprávněnými uživateli.
- **Autentičnost**, příjemce zprávy má možnost zjistit její původ, zamezení případným narušitelům vydávat se za někoho jiného.
- **Integritu**, kontrolu zprávy zda nebyla během přenosu modifikována.
- **Nepopiratelnost** odesílatel by neměl mít možnost popřít, že danou zprávu odeslal.

K zajištění uvedených služeb využívá kryptografie následujících prostředků:

- Symetrické a asymetrické šifrovací algoritmy
- Hašovací funkce
- Digitální podpis
- Kryptografické protokoly a další.

1.1 Symetrická kryptografie

Algoritmy patřící do kategorie symetrické kryptografie používají jeden tajný klíč pro obě šifrovací operace, tzn., že jediným tajným klíčem data šifrujeme i dešifrujeme. Hlavní výhoda symetrické kryptografie je rychlost, nevýhodou je však složitější distribuce klíče.



Obr. 1.1.1: Princip symetrické kryptografie

1.1.1 Algoritmus 3DES

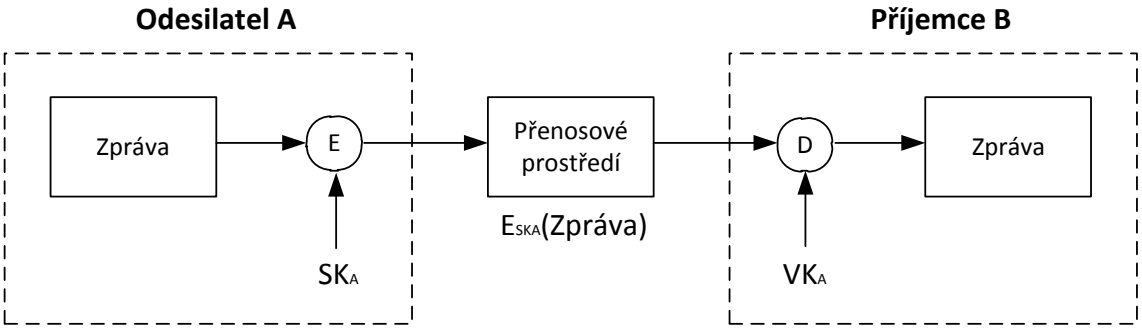
Jedná se o symetrický šifrovací algoritmus, který nahradil nedostačující 56 bitový algoritmus DES. Algoritmus 3DES není nic jiného, než 3x DES za sebou, přičemž mohou být použity 2 klíče, v tom případě se jedná o 112 bitovou variantu, nebo může být každý klíč jiný, potom se jedná o 168 bitovou variantu. Nevýhodou algoritmu 3DES je jeho rychlost, protože aplikováním tří algoritmů DES za sebou se také třikrát snížila jeho rychlost. Dnes byl již algoritmus nahrazen podstatně rychlejším a bezpečnější algoritmem AES.

1.1.2 Algoritmus AES

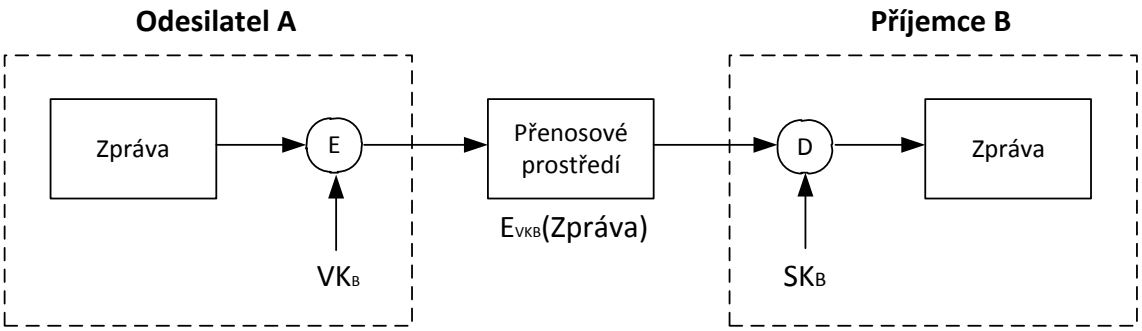
Algoritmus AES je taktéž symetrický šifrovací algoritmus, který nahradil již nedostačující algoritmus DES. Vznikl v roce 2001 po vyhlášení veřejné soutěže, kterou vyhrála šifra Rijndael, jehož tvůrci byli Joan Daemen a Vincent Rijmen. Ten byl přijat americkým Národním institutem pro standardizaci a technologie (NIST) pod názvem AES.
AES je iterovaná bloková šifra s délkou bloku 128 bitů, která používá délku klíčů 128, 192 nebo 256 bitů. AES pracuje v rundách, jejichž počet je 10, 12 nebo 14 v závislosti na délce klíče. Jedná se o velmi rychlou šifru s minimálními nároky na výpočetní výkon a paměť.

1.2 Asymetrická kryptografie

Asymetrická kryptografie se vyznačuje existencí dvou klíčů, tzv. klíčového páru. Jeden klíč je tzv. soukromý a slouží k podepisování a dešifrování dat. Druhý je tzv. veřejný a slouží k ověření podpisu a k šifrování.



Obr. 1.1.2: Proces podepisování



Obr. 1.1.3: Proces šifrování

1.2.1 Algoritmus RSA

Jedná se o asymetrický šifrovací algoritmus, který byl pojmenován po svých tvůrcích, kterými byli Ronald Rivest, Adi Shamir a Leonard Aleman. Výhodou oproti symetrickým šifrovacím algoritmům je, že řeší problém distribuce klíč ovšem za cenu značné výpočetní náročnosti. Nejlepším řešením je použití hybridního šifrování, které kombinuje obě možnosti – velkou zprávu zašifrujeme rychlou symetrickou šifrou a její klíč následně pomocí asymetrického šifrování. Při volbě délky klíče bychom měli brát v úvahu jeho využití. Klíče o délce 512 bitů byly prolomeny útoky hrubou silou již v roce 1997, proto je doporučeno používat minimálně 1024 bitů, pro větší bezpečnost pak 2048, či 4096 bitů. Nevýhodou delších klíčů je fakt, že s dvojnásobnou velikostí klíče roste doba generování zhruba 16x a doba nutná pro dešifrování a šifrování 8x resp. 4x. Alternativou RSA je DSA, který není svázán licenčními podmínkami.

1.3 Digitální otisk dat

Digitální otisk dat (haš) slouží k jednoznačnému určení vstupních dat na základě jejich digitálního otisku. Hašovací algoritmus nám umožňuje převod vstupního řetězce dat proměnlivé délky na výstupní řetězec pevné délky. Haš se používá k ukládání hesel, ke kontrole integrity souborů a také při digitálním podpisu.

Požadavek na bezpečnou hashovaní funkci je, aby byla jednocestná, tj. aby nebylo možné z haše získat zpět původní zprávu a bezkolizní. Protože haš má konečnou hodnotu např. MD5 maximálně 2^{128}kombinací, ale vstupních zpráv je nekonečně mnoho, je tedy důležité, aby neexistoval algoritmus, díky kterému by bylo možné najít dvě zprávy se stejným hašem.

Útoky za účelem získání vstupního řetězce, především přihlašovacího hesla, na základě otisku jsou následující:

- **Útok hrubou silou (Brutal-force attack)**Tento způsob je založen na testování všech možných řetězců dané délky a znakové sady. To znamená, že pokud by bylo heslo

složeno z písmen malé abecedy a čísel o délce max. 5 znaků, máme $36^1+36^2+36^3+36^4+36^5+36^6$ kombinací. Vytvoříme haš ke každé kombinaci a ten porovnáme s hašem, který chceme prolomit. Proto je dobré volit dlouhá hesla za použití všech možných znaků.

- **Slovníkový útok (Dictionary attack)**Tento způsob je založen na tom, že máme slovník se slovy, ke kterým jsou přiřazeny haše, útok pak jako v předešlém případě porovnává haš s tou ve slovníku. Tento způsob je ovšem spíše teoretický, protože spousta aplikací zabraňuje použití známého slova jako hesla, nebo na to alespoň upozorní.
- **Rainbow tables** V tomto případě máme předgenerovanou tabulku hašů k daným heslům, ve které se potom vyhledává. Nevýhodou je ovšem značná velikost tabulky (až několik desítek GB).

Jedním z možných způsobů, jak se chránit před slovníkovými útoky, je použití tzv. „solení hašů“. To znamená, že se na vstup k heslu přidá ještě náhodný řetězec, který je pokaždé jiný, což má za následek, že dvě stejná hesla mají dva rozdílné haše, a tak při prolomení hesla jednoho uživatele nejsou ohroženi ostatní, kteří použili stejné heslo.

1.3.1 MD5

MD5 (Message Digest Algorithm 5) je kryptografická hašovací funkce, která vrací digitální otisk ze vstupu o velikosti 128 bitů. V dnešní době se již nepoužívá. Protože v roce 2004 byly objeveny zásadní chyby v jejím návrhu čínským týmem Dr. Wangové a později českým kryptologem Vlastimilem Klímou, který objevil algoritmus, díky kterému je možné nalézt kolizi zhruba za 2 minuty.

1.3.2 SHA

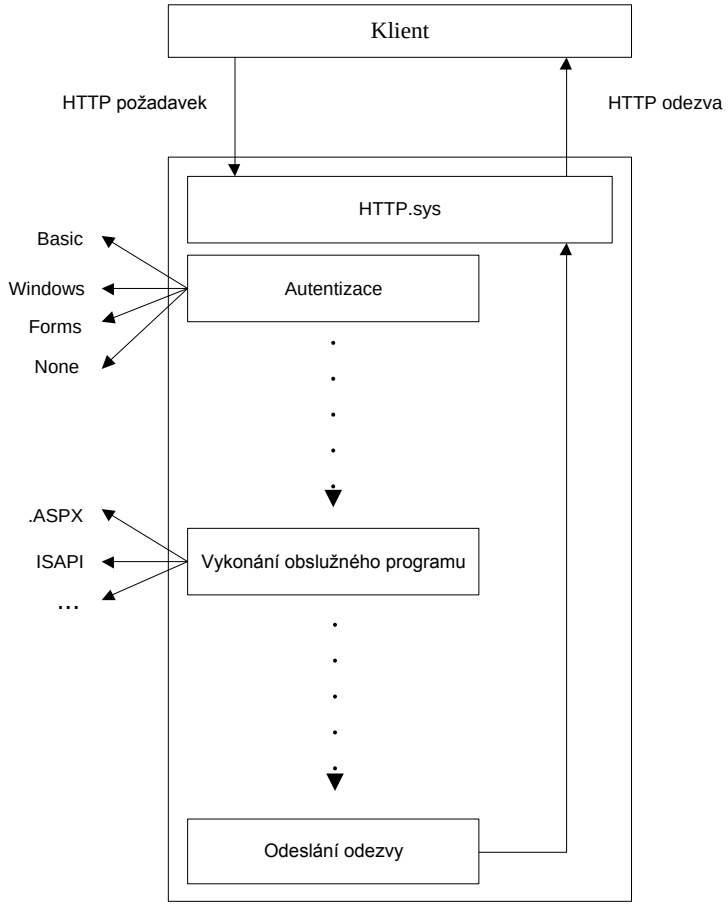
SHA (Secure Hash Algorithm) je kryptografická hashovací funkce, která vrací digitální otisk ze vstupu o velikosti 160 bitů, tato varianta se označuje jako SHA-1, nebo o velikosti 256 až 512 bitů a ta se označuje jako SHA-2. Algoritmus SHA je využit v bezpečnostních aplikacích a protokolech, jako jsou např. TLS, SSL, PGP, SSH a IPSec. U algoritmu SHA-1 byla také nalezena kolize, ovšem v tomto případě není problém závažný, jelikož její nalezení je časově náročné a trvá 1-2 měsíce.S současné době se připravuje nový algoritmus SHA-3, která nebude vázána žádnými autorskými právy a bude schopna po desetiletí udržet v bezpečí citlivé informace. Algoritmus SHA-3 by měl mít stejnou výstupní velikost jako SHA-2, ale měl by obsahovat určitá bezpečnostní a výkonová vylepšení (viz literatura[5]).

Tab. 1.3.1 Porovnání jednotlivých verzí algoritmů SHA

Algoritmus	Výstupní velikost (bit)	Velikost bloku (bit)	Max. velikost zprávy (bit)	Délka slova (bit)	Kolize
SHA-0	160	160	2^{64}-1	32	Ano
SHA-1	160	160	2^{64}-1	32	Ano
SHA-256/224	256/224	512	2^{64}-1	32	Ne
SHA-512/384	512/384	1024	2^{128}-1	64	Ne

2 WEBOVÝ SERVER IIS

Internetová informační služba (IIS) je webový server společnosti Microsoft, který slouží k publikování webových stránek na internetu. Webový server má modulární strukturu a jeho funkce lze rozšířit pomocí rozšiřovacích modulů.



Obr. 2.1: Blokové schéma IIS 7.0

V současné době je možné se setkat se dvěmi verzemi tohoto serveru, které se výrazně liší svoji architekturou. Verze IIS 6.0 je postavena na rozšiřujících modulech ISAPI, které předávají požadavky vyšší vrstvě, která je následně zpracuje. Tvoří tedy mezivrstvu. Novější verze IIS 7.0 umožňuje použití rozšiřujících HTTP modulů, které mohou přímo zpracovat danou událost. Tato verze je tedy plně modulární, nabízí unifikovaný konfigurační systém a lepší integraci technologie ASP.NET (viz obr. 2.1).

3 PLATFORMA.NET FRAMEWORK A ASP.NET

Obsahem této kapitoly je seznámení s platformou .NET Framework a její struktury, dále se kapitola věnuje programovacímu jazyku ASP.NET. (viz lit. [1], [2])

3.1 .NET Framework

.NET Framework je platforma vytvořená společností Microsoft a slouží k zjednodušenému vývoji aplikací. Tato platforma zajišťuje:

- Platforma poskytuje běhové prostředí pro vykonání kódu
- Řeší správu a navrácení systémových zdrojů
- zajišťuje jazykovou nezávislost

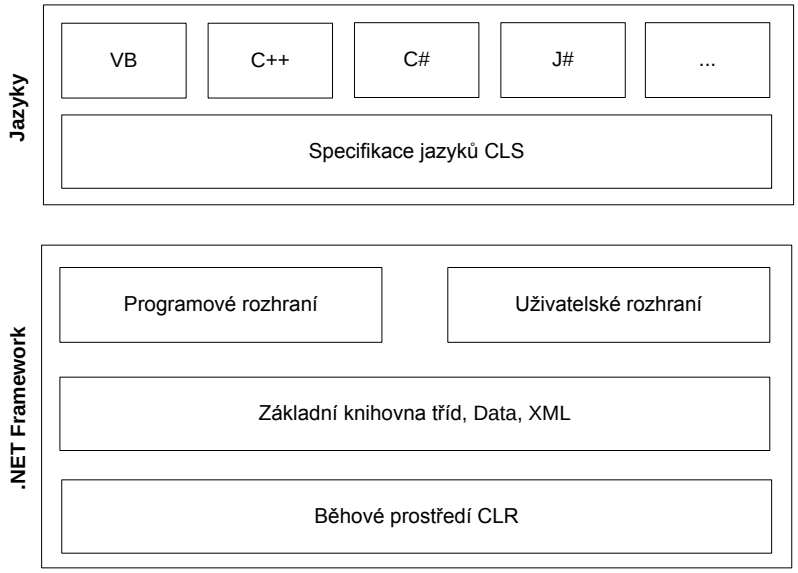
3.1.1 Struktura .NET Framework

Strukturu .NET Frameworku tvoří několik vrstev:

- **CRL** (Common Language Runtime) jedná se o základ celého frameworku. umožňuje správu paměti a řízení životního cyklu aplikace (Garbage collector), dále překladač z jazyka MSIL (Microsoft Intermediate Language) do nativního kódu a dále služby, které zapouzdřují Win32 funkce.

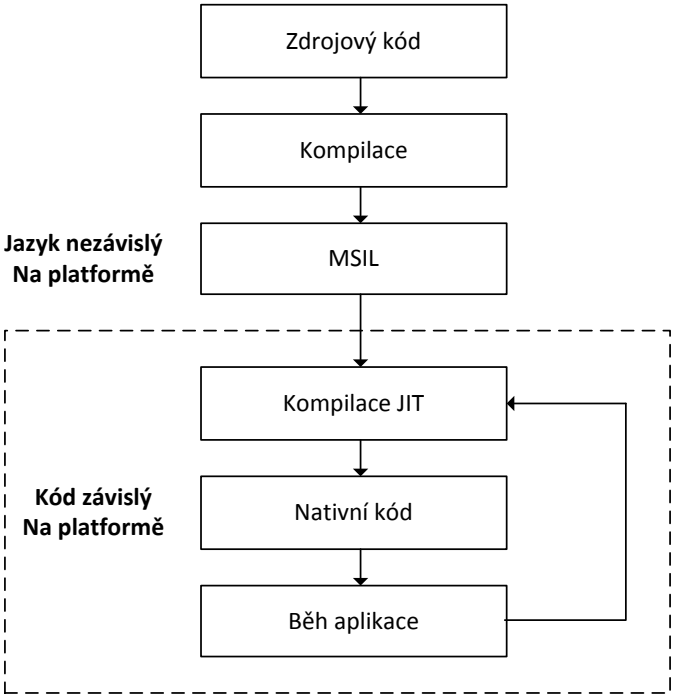
- **Základní knihovna tříd**, která má na starosti přístup k datům a práci s XML.

- **Vyšší vrstvy**, které mají na starosti tvorbu uživatelského rozhraní, webových aplikací apod.



Obr. 3.1.1: Schéma .NET Frameworku

Jelikož je tato platfroma jazykově a platformou nezávislá, nedochází ke kompilaci kódu přímo. Nejprve se kód zkompiluje do jazyka MSIL, který je nezávislý na architektuře procesoru, to znamená, že může běžet na jakémkoli systému s běhovým prostředím CLR. Následně dochází ke druhé kompilaci, ke které dochází před samotným spuštěním aplikace. Tato kompilace se nazývá JIT (Just in Time). Tato kompilace proběhne jen jednou a to při prvním spuštění aplikace, je závislá na architektuře. Schéma kompilace zobrazuje obr. 3.1.2:



Obr. 3.1.2: Zpracování zdrojového kódu

3.1.2 Verze .NET Frameworku

- **Microsoft .NET Framework 1.0** byla první verze, která spatřila světlo světa v roce 2002.

- **Microsoft .NET Framework 1.1**byl vydán v roce 2003 a sloužil jako servisní balíček pro předešlou verzi, nepřinesl žádné výrazné změny, ale spíše opravy. Verze frameworku 1.0 a 1.1 byly mezi sebou nekompatibilní.

- **Microsoft .NET Frameworku 2.0**byl vydán v roce 2005 a přinesl spoustu koncepčních změn. Tato verze byla také doplněna o nové třídy. K této verzi Frameworku vzniklo několik rozšíření, např. Ajax.

- **Microsoft .NET Framework 3.0** byl vydán v roce 2006 a přinesl sadu rozšiřujících knihoven pro předešlou verzi a dále technologie:

  - *Microsoft CradSpace*je systém určený k vytváření vztahů mezi weby a online službami.

  - *WPF (Windows Presentation Foundation)* je technologie pro vytvoření uživatelsky bohatého rozhraní.

  - *WWF (Windows Workflow Foundation)*je nové rozhraní pro správu, vývoj a modelování aplikací

  - *WCF (Windows Communication Foundation)* je technologie pro vývoj distribuovaných aplikací na libovolném komunikačním protokolu.

- **Microsoft .NET Framework 3.5** byl vydán v roce 2008 a navazuje na předešlou verzi. Rozšířil jazyky C# 3.0 a Visual Basic 9.0. Dále přinesl podporu pro LINQ (Language Integrated Query) a rozšíření pro WCF a WWF. Optimalizoval také síťové aplikační rozhraní.

- **Microsoft .NET Framework 4.0** je novou verzí, která přináší mnoho změn v konceptu. umožňuje lepší integraci technologií, přinesl nové datové typy (např. BigInteger), dynamické proměnné a podporu pro paralelní programování.

3.1.3 Jmenné prostory .NET Frameworku

.NET Framework obsahuje mnohé jmenné prostory, což jsou třídy, které mají na sebe logickou návaznost. Nyní některé z nich jsou uvedeny níže:

- **System** obsahuje třídy, které mají na starosti základní funkce.
- **System.IO** obsahuje třídy pro vstupně/výstupní operace.
- **System.NET** obsahuje třídy pro práci s mnoha sítovými protokoly.
- **System.Collections** obsahuje třídy pro práci s kolekcemi dat, jako jsou např. struktury, seznamy a nebo hašovací tabulky.
- **System.Data** obsahuje třídy pro práci s technologií ADO.NET
- **System.Web** obsahuje třídy, které umožňují přístup k protokolu HTTP. umožňují zpracovávat požadavky, odezvy a dále obsahují nejrůznější nástroje pro práci s protokolem HTTP.
- **System.Xml** obsahuje třídy pro práci se soubory XML.
- **System Security** obsahuje třídy, které mají na starosti autentizaci, autorizaci a kryptografické funkce
- **System.Net.Security** obsahuje třídy pro zajištění bezpečné komunikace bezi koncovými body. zajišťuje autentizaci a dále komunikaci na bázi SSL.

3.2 Jazyk ASP.NET

Je nový jazyk, který je nástupcem skriptovacího jazyka ASP. Pracuje na zcela odlišných principech, aplikace založené na ASP.NET jsou totiž na rozdíl od skriptovacích jazyků, jako jsou např. ASP či PHP překompilovány do DLL souborů, což s sebou přináší vyšší výkon i stabilitu. Jazyk ASP.NET umožňuje:

- Objektově orientované programování
- Návaznost na .NET Framework umožňuje psát zdrojový kód v jakémkoli jazyce (např. C#, VB.NET)
- umožňuje oddělit kód stránky od kódu programu
- Využití serverových ovládacích prvků a webových formulářů pro jednoduchý a rychlý vývoj.

- umožňuje využít jmenné prostory .NET Frameworku, odpadá tak zásadní rozdíl mezi vývojem desktopových a webových aplikací.

3.2.1 Aplikace ASP.NET

U tradičních desktopových aplikací se aplikace skládají ze spustitelných souborů s příponou EXE a jim přidružených dynamických knihoven s příponou DLL. U webového programování je tomu ale zcela jinak. Webové aplikace ASP.NET se skládají ze stránek a z ovladačů, obslužných rutin (tzv. handlerů), modulů a spustitelných kódů, které se dají vyvolat z virtuálního adresáře na webovém serveru.

3.3 Správa stavu v ASP.NET

Protokol HTTP je bezstavový, to znamená, že webový server neudržuje trvalé spojení s klienty, což má za následek to, že nedokáže rozpoznat jejich požadavky. Tento problém umožňuje překlenout správa stavových objektů, která je součástí ASP.NET. Ta se dělí na klientské a serverové. (viz lit [1], [2]]

3.3.1 Klientské stavové objekty

Klientské stavové objekty umožňují uložit informace o stavu webové aplikace na straně klienta. Jejich nevýhodou je, že mohou uchovat jen malé množství informací a mohou být vystaveny bezpečnostním rizikům.

- **Cookies** umožňují uložení malého množství dat na klientském počítači. Maximální velikost jedné cookie jsou 4kB. Dalším omezením je jejich počet, který činí 20 cookies na jednu doménu. Cookie mohou představovat bezpečnostní riziko a tak se nehodí pro ukládání citlivých informací, jako je např. heslo.

- **Query Strings** se nejčastěji využívá, přesouváme-li malé množství z jedné stránky na druhou.

- **Hidden Fields** slouží k uložení malého množství dat. Ačkoli se jedná o skrytá pole, je možné je číst a tak mohou představovat bezpečnostní hrozbu v případě, jsou-li v ní uloženy citlivé informace.

- **View State** slouží k uložení malého množství dat. Nevýhodou je ovšem vysoká režie, kdy přenášíme přes internet poměrně malé stránky a velké množství View State dat. Tento problém byl částečně kompenzován v novém .NET Frameworku 4.0, ve kterém je možné zapnout View State pouze pro daný ovládací prvek.

3.3.2 Serverové stavové objekty

Serverové stavové objekty jsou uloženy na serveru, nabízejí vyšší míru bezpečnosti, ale zabírají více zdrojů serveru.

- **Application** je slouží k uložení občasných změn a globálních informací, které využívá mnoho uživatelů. Dané řešení je vhodné pouze pro uložení malého množství dat a neposkytuje vysokou míru zabezpečení.

- **Profile Properities** slouží k uložení uživatelských dat, která přetrvávají, i když session vyprší.

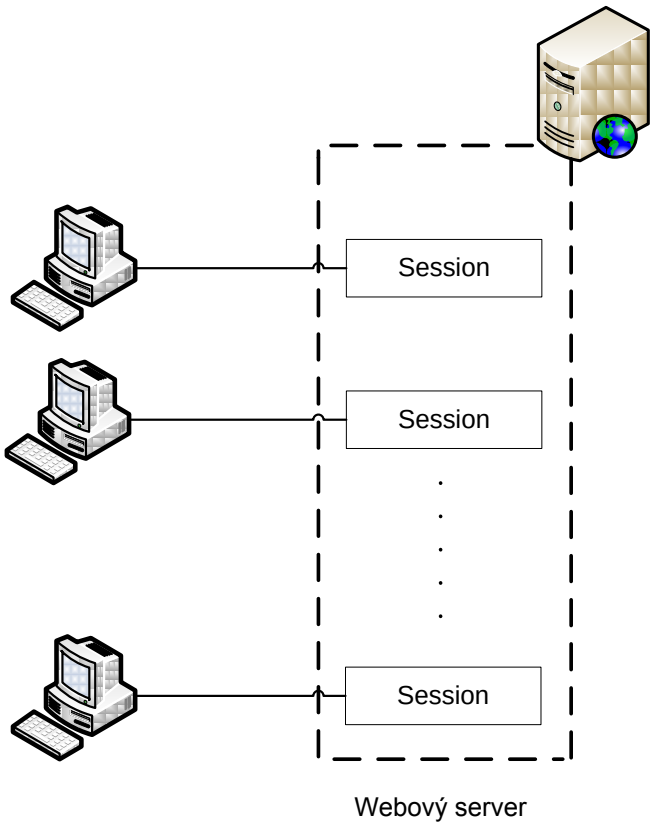
- **Session State** identifikuje požadavky z daného webového prohlížeče a ukládá uživatelská data na straně serveru. Výhodou daného řešení je, že funguje i na webových pohlížečích, které nepodporují HTTP cookies, nicméně se spolu s cookies velmi často používá, jelikož bez podpory cookies je session ID uloženo v query string, který tvoří potenciální bezpečnostní riziko.

- **Database** slouží k uložení velkého množství dat do databáze. Dané řešení je velmi robustní a nabízí vysokou míru bezpečnosti.

Nyní se budeme podrobněji věnovat session ASP.NET.

3.3.3 Session ASP.NET

Technologie session ASP.NET (viz lit. [6],[7],[8]) slouží k překlenutí bezstavovosti protokolu HTTP. Data jednotlivých klientů jsou uložena odděleně v paměti serveru.K identifikaci dat jednotlivých klientů je použito 120 bitového identifikátoru.



Obr. 3.3.1: Princip Session ASP.NET

Existují tři možnosti uložení uživatelských dat:

- **In State** varianta umožňuje uložit data v paměti serveru, tato varianta je nejsnazší a poskytuje nevyšší výkon. Nevýhodou je jsou však velké nároky na paměť serveru a fakt, že restart webové služby má za následek ztracení všech session.

- **State Server** varianta umožňuje uložit data na speciální server, výhodou daného řešení je, že restart služby nikterak neovlivní uložené session. Nevýhodou je však složitější implementace, jelikož je nutné data serializovat a deserializovat, což má také za následek nižší výkon.Další nevýhodou je fakt, že state server není možné replikovat a tak při použití serverové farmy tvoří úzké hradlo, jelikož k němu přistupují všechny servery, taktéž výpadek má za následek ochromení celé serverové farmy. Toto řešení je spolu s SQL Serverem nejvhodnější pro použití ve serverových farmách.

- **SQL Server**je nejrobustnější varianta je podobná variantě State Server s tím rozdílem, že jsou data uložena v SQL serveru, jehož výhoda spočívá v tom, že jej lze na rozdíl od state serveru replikovat.

4 AUTENTIZACE

Autentizace je jedna ze základních funkcí, kterou lze zabezpečit pomocí moderních kryptografických prostředků. Umožňuje vzájemné ověření dvou komunikujících stran. (viz lit [1], [2], [3])

4.1 Autentizace v IIS

Internetová informační služba (IIS) využívá 5 způsobů autentizace, které jsou uvedeny níže a budou blíže rozvedeny.

- **Basic** autentizace je jeden ze způsobů HTTP autentizace popsaný v RFC 1945 jako součást protokolu HTTP 1.0 a později jako RFC 2616 jako součást protokolu HTTP 1.1.Basic autentizace umožňuje ověření uživatele prostřednictvím uživatelského jména a hesla. Pakliže chceme ověřit klienta, server odešle odezvu s autentizačním záhlavím, které má následující tvar:

  WWW-Authenticate: Basic realm="Secure Area"

  Webový prohlížeč klienta na danou událost odpoví výzvou pro zadání přihlašovacích údajů. Výzva zobrazí také realm, což je zpráva pro klienta, nejčastěji slouží jako identifikace serveru. Přihlašovací údaje jsou následně jako součást nového autentizačního záhlaví v kódu Base 64odeslány na server,nejedná se tedy o bezpečné řešení. Odezva klienta má následující formát:

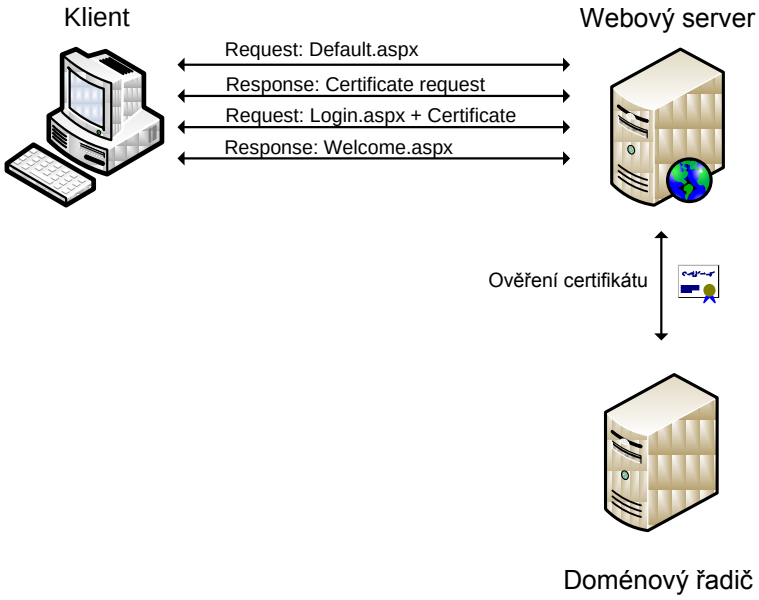
  Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

- **Digest** autentizace je součástí standardu HTTP 1.1. Je podobná basic autentizaci, ale s tím rozdílem, že pro zabezpečení uživatelského hesla využívá hašovací funkci MD5. Implementace této autentizační metody je v systému Windows ovšem modifikovaná, a tak ji lze bez potíží zprovoznit pouze v prohlížeči Internet Explorer.

- **Integrated** autentizace využívá NTLM, či Kerberos autentizaci uživatelů v doméně, či Active Directory. Na rozdíl od Basic a Digest autentizace neposílá heslo přes síť, což z ní dělá velmi bezpečnou autentizační metodu.

- **Certificate Mapping**je nejbezpečnější autentizační metoda, která využívá k autentizaci digitální certifikáty, které jsou nainstalovány na počítači. Pokud se počítač pokusí připojit k serveru, digitální certifikát automaticky autentizuje uživatele. Pokud

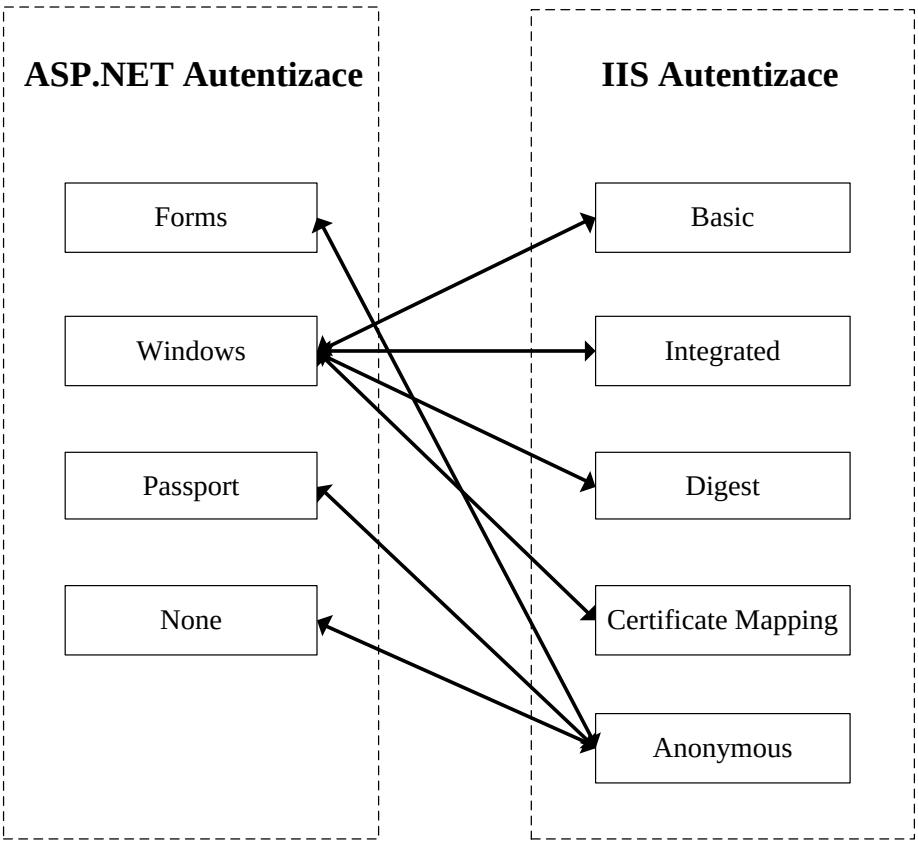
nastavíme Windows Authentication provider v ASP.NET, vlákno aplikace poběží pod uživatelem, pro kterého byl daný certifikát vydán.



Obr. 4.1.1: Autentizační metoda Certficate Mapping

- **Anonymous** autentizace je využívána v případě, není-li potřeba autentizovat uživatele přistupující k webové stránce.

Autentizační metody IIS jsou propojeny s autentizačními metodami ASP.NET. Toto propojení znázorňuje obr. 4.1.2.

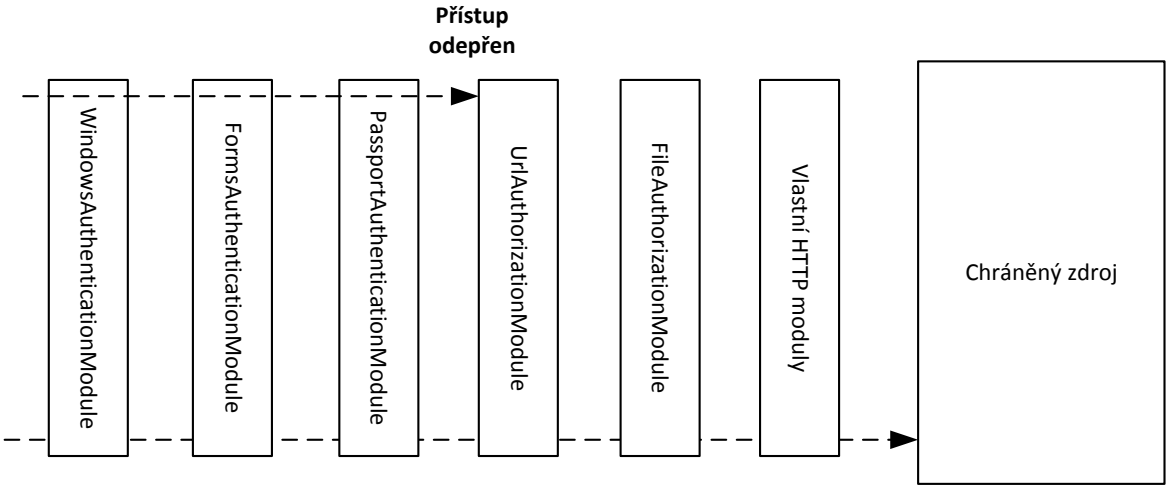


Obr. 4.1.2: Možnosti Autentizace IIS a ASP.NET

4.2 Autentizace v ASP.NET

Autentizace v ASP.NET je implementována pomocí speciálních HTTP modulů (viz obr. 4.2.1), které zajišťují danou formu autentizace. ASP.NET poskytuje 3 základní HTTP moduly sloužící pro následující způsoby autentizaci:

- Formulářová autentizace
- Windows autentizace
- Passport autentizace



Obr. 4.2.1: Princip HTTP modulů sloužících pro autentizaci a autorizaci

4.2.1 Formulářová autentizace

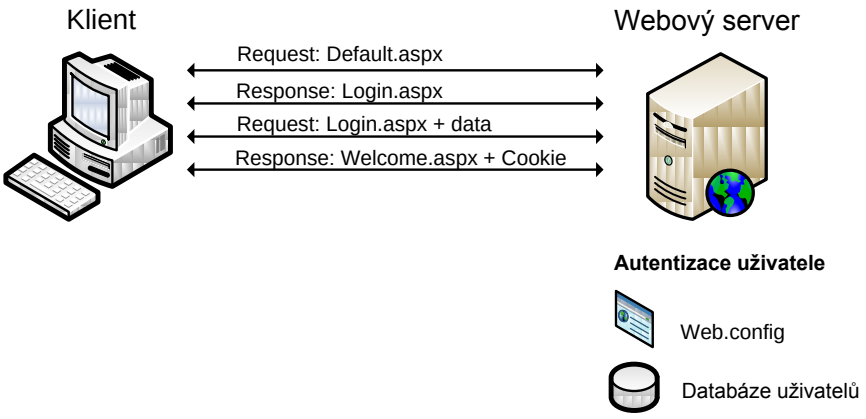
Formulářová autentizace odkazuje na volitelnou komponentu uživatelského rozhraní, která akceptuje uživatelské pověření, např. uživatelské jméno a heslo. Jakmile je uživatel autentizován, obdrží pověření (tiket), který indikuje, že uživatel byl autentizován pro dlaší požadavky. Toto pověření se skládá z několika polí (viz lit. [19]):

- **Verze** obsahuje číslo verze
- **Jméno** obsahuje jméno uživatele asociovaného s pověřením.
- **Datum vystavení** obsahuje čas, kdy bylo pověření vystaveno
- **Expirace** obsahuje čas, kdy pověření vyprší
- **Délka trvání** obsahuje logickou hodnotu True, pakliže je pověření trvalé, v opačném případě obsahuje hodnotu False
- **Uživatelská data** toto pole obsahuje uživatelem definovaná data, mohou být libovolná.

Toto pověření může být distribuováno dvěma způsoby:

- **Pomocí cookie** využívá pro přenos pověření cookie. Cookie je malé množství dat, které nabízí server klientovi. Tato data klient pošle serveru při každém http požadavku. Je však nutné respektovat omezení, které s sebou cookie nese, tím je především jeho maximální velikost, která činní 4kB.

- **Pomocí URL** tato metoda je obsažena v ASP.NET 2.0 a umožňuje přenášet pověření jako součást URL adresy. Tato metoda je terčem mnoha útoků, které mají za cíl krádež identity.



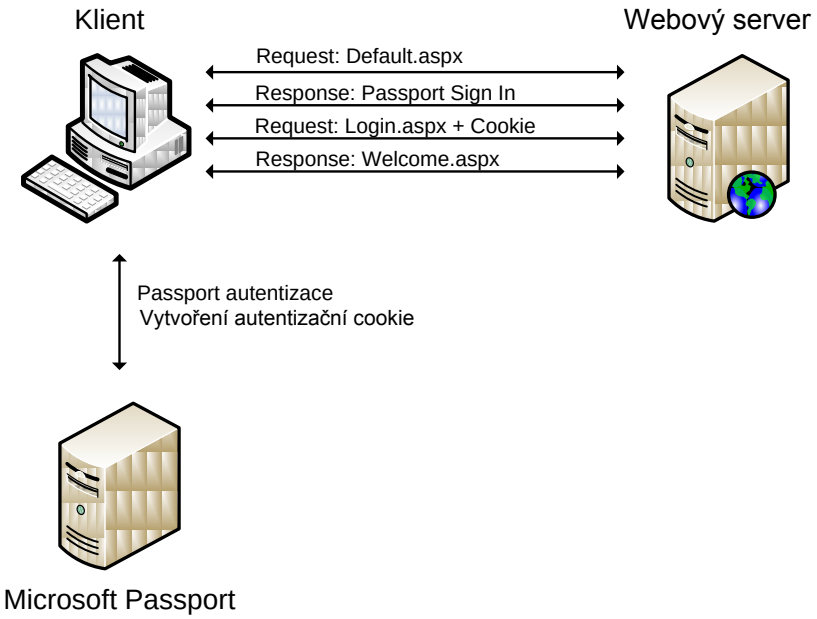
Obr. 4.2.2: Autentizace prostřednictvím Forms

4.2.2 Windows autentizace

Tento způsob autentizace využívá autentizaci IIS, která je následně mapována na uživatelské účty Windows. Její výhoda spočívá v snadné implementaci a možnosti využití neviditelného režimu, kdy je uživatel automaticky přihlášen na základě systémového účtu Windows. Způsoby autentizace jsou blíže rozepsány v kapitole 4.1.

4.2.3 Passport Autentizace

Tento způsob autentizace je založen na centralizované autentizační služba poskytované firmou Microsoft, informace o uživatelích jsou uloženy na serveru společnosti Microsoft. Jelikož danou službu doprovázelo mnoho bezpečnostních problémů, nebyla příliš oblíbená a velké firmy od ní opouštěly. Tato metoda autentizace však dnes již není příliš využívána.



Obr. 4.2.3: Autentizace prostřednictvím služby Microsoft Passport

4.3 Zhodnocení autentizačních metod

Výše uvedené autentizační metody mají své kladné i záporné stránky, stejně tak i rozdílné možnosti využití. Některé jsou vhodné pro autentizaci uživatelů systému Windows, ty naleznou uplatnění především v intranetových webových aplikacích, jiné využívají jako úložiště databáze. Takové řešení je pak vhodné spíše pro internetové webové aplikace. Výhody a nevýhody jednotlivých metod shrnuje tabulka XX.

Tab.4.3.1: Zhodnocení autentizačních metod ASP.NET

Autentizační metoda	**Výhody**	**Nevýhody**
Formulářová autentizace	• Možnost realizace vlastního webového rozhraní • Kontrola nad autentizačním kódem • Volitelný způsob uložení uživatelských informací	• Nebezpečí odposlechu komunikace, nutnost použít SSL • Nutnost správy pověření
Windows autentizace	• Spojeno s účty Windows • Autentizace IIS, odpadá	• Spojeno s účty Windows (může být považováno také za

	tak správa pověření • Snadná implementace	nevýhodu) • Nemožnost definice vlastního webového rozhraní • Některé formy autentizace IIS fungují spolehlivě pouze v Internet Exploreru
Passport Autentizace	• Správa je za poplatek v rukou Microsoftu.	• Odstranění bezpečnostních rizik Microsoftem (může trvat déle) • Nepříliš rozšířené, tedy nejistá budoucnost

5 ÚTOKY NA WEBOVÉ SLUŽBY

Webové aplikace mohou obsahovat bezpečnostní chyby, které lze zneužít za účelem odcizení identity a nebo přístupu k citlivým údajům. V následujícím textu budou jednotlivé metody blíže rozebrány. Viz lit ([10], [11], [18])

5.1 SQL Injection

Jedná se o způsob napadení databáze vložením kódu prostřednictvím neošetřeného vstupu. Cílem je provedení útočníkem modifikovaného dotazu, který může získat, či pozměnit data uložená v databázi. Útok spočívá v tom, že se hodnoty během zadávání SQL příkazu zadávají do apostrofů. Mějme např. dotaz

SELECT * FROM Users WHERE login = '"+login+" ' AND pwd = '"+pwd+" '

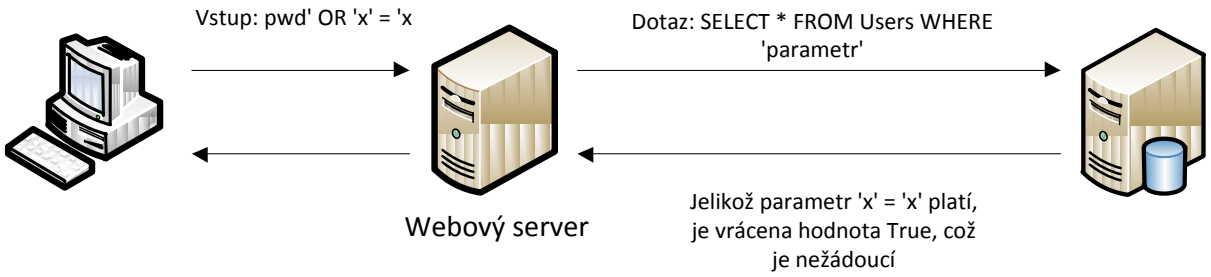
Pakliže by útočník zadal jako uživatelské jméno Admin a heslo ve tvaru pwd' OR 'x' = 'x, dostaneme dotaz

SELECT * FROM Users WHERE login = 'Admin' AND pwd = 'pwd' OR 'x' = 'x'

Jelikož 'x' = 'x' platí vždy, je útočník přihlášen jako pod administrátorským účtem. Toto je ovšem nežádoucí a je nutné se proti takovému počínání bránit. To je možné několika způsoby:

- **Použití ověřovacích prvků** spočívá v ověření vstupů z formulářových prvků dle daného vzoru. K tomu je v ASP.NET možné využít ověřovacího prvku RegularExpressionValidator a RangeValidator. Další možností je využít funkci Regex.IsMatch, která porovnává vstupní řetězec s daným vzorem.

- **Použití parametrizovaných dotazů** je další varianta ochrany před útoky typu SQL Injection. Spočívá ve využití parametrizovaných vstupů, které jsou před vykonáním dotazy automaticky ošetřeny tak,a by neobsahovaly nebezpečné znaky. Tato metoda je dnes nejpoužívanější.



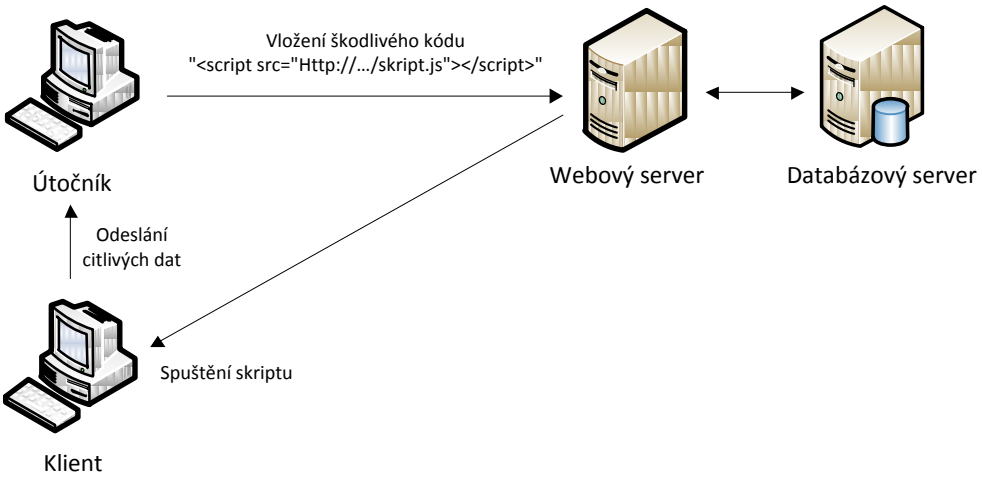
Obr. 5.1.1: SQL Injection

5.2 Cross-Site Scripting (XSS)

Je druh útoku, který spočívá ve vložení nebezpečného kódu v klientském skriptovacím jazyce, jako je Java script, VB script apod. do webových stránek. Rozeznáváme 3 druhy útoků:

- **Okamžitý útok** je útok, který je vykonán ihned po vykonání nebezpečného skriptu. Tento útok zneužívá např. parametrizovaných URL, které jsou podstrčeny uživateli, v takovém případě hrozí nebezpečí zcizení citlivých informací.

- **Perzistentní útok** spočívá v možnosti vložení vlastního obsahu na web (např. do diskusního fóra). Neošetřené vstupy tak umožní vložení nebezpečných skriptů. Pakliže pak uživatel přistoupí na takovou stránku, může se stát obětí škodlivého skriptu.



Obr. 5.2.1: Princip perzistentního XSS útoku

- **DOM-based útok** spočívá v tom, že škodlivý skript není uložen na webový server, nýbrž je spuštěn z počítače uživatele. Dochází tak ke zneužití lokálních webových aplikací, což s sebou přináší riziko, jelikož lokálně spuštěné skripty bývají považovány za bezpečné.

5.3 Session hijacking

Sesion hijacking je metoda, která umožňuje získat kontrolu nad relací a následným získáním kontroly nad webovou aplikací uživatele.

Existují 3 způsoby útoku:

- **Brute Force** útok spočívá v tom, že se útočník pokouší odhadnout session ID, dokud neuspěje.

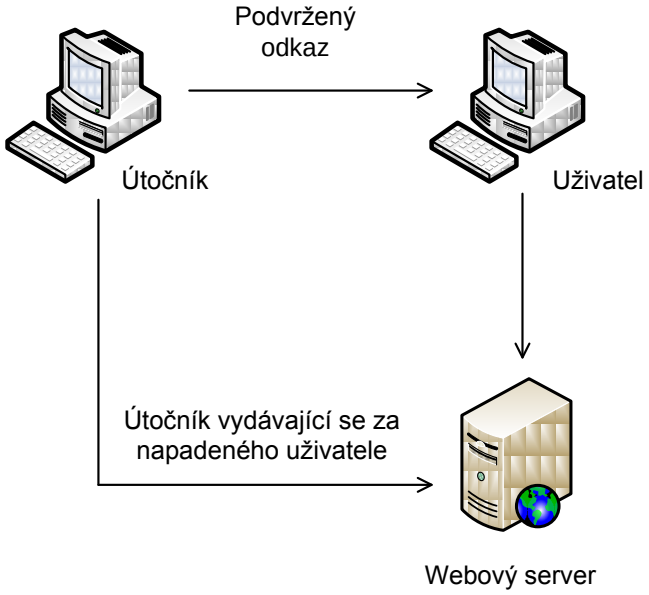
- **Calculate** útok spočítá v tom, že session ID není zcela náhodně generováno a lze jej spočítat.

- **Steal** útok používá odlišné techniky, útočník se pokouší získat session ID.

V případě, že spojení není šifrováno pomocí SSL je Session ID posíláno přes síť v otevřené podobě a je možné jej získat a tím útočníkovi usnadnit práci.

5.4 Session fixation

Tento útok spočívá v tom, že je uživateli podstrčeno předem vygenerované Session ID, jakmile se uživatel přihlásí, server považuje tento identifikátor za session ID uživatele. Pomocí tohoto session ID se pak může přihlásit i útočník.



Obr. 5.4.1: Princip session fixation

5.5 Zhodnocení bezpečnostních rizik

Výše uvedené útoky jsou převážně směřované na session ID, které se pokoušejí odcizit a nebo odhadnout. Proti těmto útokům se lze částečně bránit, nicméně jedinou spolehlivou metodou je použití autentizace bez cookies. V případě použití cookies je vhodné šifrovat komunikaci za pomoci technologie SSL, při které není možné odposlouchávat komunikaci mezi klientem a serverem. Shrnutí jednotlivých útoků zobrazuje tab. 5.4.1.

Tab. 5.4.1: Shrnutí útoků na webové služby

Typ útoku	**Možnost obrany**
SQL Injection	Nejbezpečnější metodou obrany je použití parametrizovaných dotazů, které není možné zneužít pro útok.
Cross-Site Scriptin	Obranou proti danému útoku je kontrola vstupu formulářů, do kterých by útočník mohl vložit škodlivý kód.
Session Hijacking	Obranou proti danému útoku je:

	• Dostatečně dlouhé a náhodné session ID, které není útočník schopen odhadnout a zneužít k útoku. Za předpokladu, že je session ID dostatečně dlouhé a omezenou dobou platnosti je téměř nemožné jej odhadnout. • Použití šifrovaného spojení SSL, které útočníkovi znemožní odposlouchávání komunikace. Nevýhodou daného řešení jsou výpočetní nároky kladené na webový server, nicméně tento nedluh lze eliminovat použitím SSL akcelerátoru. • Zabránění skiptům, aby měly přístup k cookies, nesoucích session ID. To je možné zajistit u cookies příznakem HttpOnly.
Session Fixation	Možnou obranou proti danému útoku je kontrola původu cookies. Tím je zajištěna souvislost session ID s danou IP adresou. Nevýhodou daného řešení je, že klienti sdílející internetové připojení přes NAT mají stejnou IP adresu.

6 NÁVRH A REALIZACE AUTENTIZAČNÍ METODY

Následující kapitola se bude věnovat samotné realizaci autentizační metody. Nejprve bude rozebrán její návrh, který přejde v její realizaci v prostředí .NET

6.1 Návrh autentizační metody

Při návrhu autentizační metody bude brán ohled na maximální bezpečnost a zároveň uživatelské rozhraní, které by bylo možné snadno implementovat do stávajícího systému webových stránek. Za nejbezpečnější metodu byla zvolena http autentizace v kombinaci s formulářovým přihlašováním. Ta v sobě spojuje maximální bezpečnost, jelikož odpadá nutnost používat cookies, které jsou cílem útoků a zároveň nastavitelné uživatelské rozhraní. Tu se bohužel nepodařilo z technických důvodů realizovat a tak bylo nutné využít pro autentizaci cookies. Jelikož by daná metoda nebyla zcela bezpečná, byly implementovány

metody pro zvýšení bezpečnosti.Prostředky pro realizaci výše uvedených požadavků jsou použití autentizačního HTTP modulu pro tvorbu vlastního způsobu autentizace a realizace vlastních poskytovatelů, kteří budou spravovat uživatelské účty v SQL databázi.

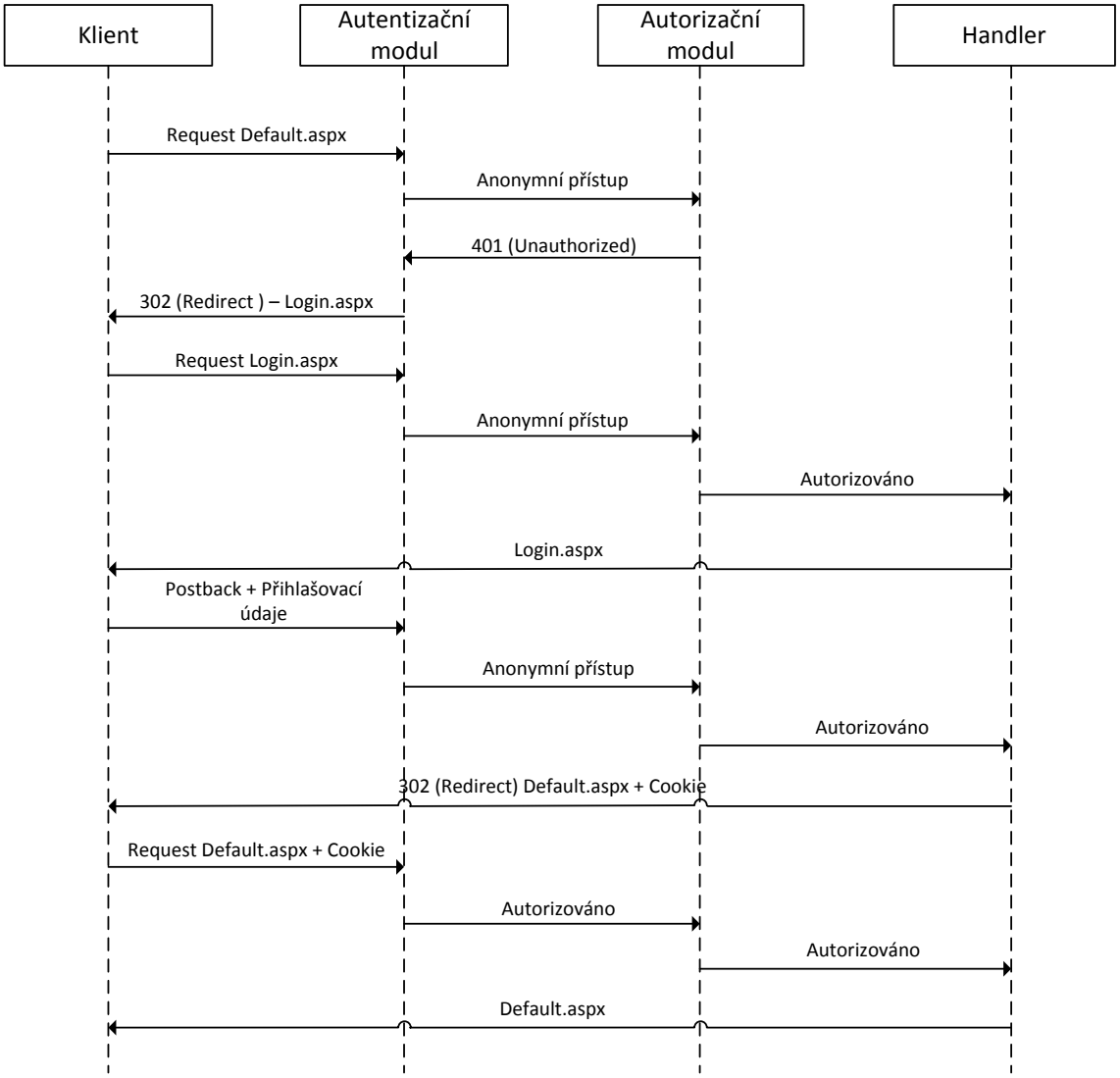
6.1.1 Autentizační modul

Tato podkapitola se bude věnovat možnostem a použití autentiačních modulů (viz lit. [15], [16]). Autentizační modul a jedním z typu http modulů, který umožňuje obsluhu požadavků na autentizaci.http modul slouží jako filtr pro příchozí požadavky na webový server a jejich odpovědi. umožňuje tak řešit autentizaci, autorizaci, přesměrovaní a jiné požadavky spojené s webovými službami. Jádrem modulu je třída, která implementuje rozhraní *Sytem.web.IhttpModule*. Tento modul je následně zaregistrován, aby mohl přijímat události. Tyto události zapojuje modul prostřednictvím metody *Init*. Tato metoda umožňuje definici následujících aplikačních událostí:

- **BeginRequest** je první z aplikačních událostí. Nastane ihned po přijetí požadavku na webový server, proto se nejčastěji využívá k realizaci přesměrování.
- **AuthenticateRequest** je aplikační událost, která nastane, je-li požadována autentizace. Umožnuje ověření příchozích požadavků vůči nastavenému poskytovateli členství.
- **PostAuthenticateRequest** je aplikační událost, která nastane ihned po autentizaci uživatele, který ještě nebyl autorizován.
- **AuthorizeRequest**je aplikační událost, jenž zpracovává požadavky na autorizaci uživatele.
- **ResolveRequestCache** je aplikační událost, která umožňuje práci s cache. Nejčastěji je využívána ke zjištění, zda-li je požadovaná stránka přítomná v cache.
- **AcquireRequestState** je aplikační událost, která umožňuje autentizovaným požadavkům přístup k session. Nejčastěji je využívána pro zápis hodnot do session, které je možné později využít.
- **PreRequestHandlerExecute** je aplikační událost, která nastane těsně před tím, než jsou požadavky vykonány obslužnou rutinou.
- Nyní dochází k vykonání kódu stránky.
- **PostRequestHandlerExecute** je aplikační událost, která nastane v bezprostředním okamžiku po vykonání obslužné rutiny.

- **ReleaseRequestState** je aplikační událost, která slouží pro přístup k session v okamžiku, jsou li data vykonána handlerem. Nejčastěji je využíván pro uložení modifikovaných dat zpět do session.
- **UpdateRequestCache** je aplikační událost, která slouží k uložený potřebných dat do cache.
- **EndRequest** je aplikační událost, která je vykonána bezprostředně před tím, než je odeslána odpověď klientovi.

V navrhovaném modulu, který bude sloužit pro autentizaci budou využité aplikační události BeginRequest, která bude sloužit k přesměrovaní na bezpečnou stránku šifrovanou pomocí technologie SSL a dále pro získání potřebných informací z přihlašovacího formuláře. Tyto informace budou následně zapouzdřeny do autentizační hlavičky a předány dalším aplikačním událostem ke zpracování. Dalším použitým modulem je AuthenticateRequest, který bude jádrem autentizačního modulu. Umožňuje příjem a dekódování autentizační hlavičky, získané přihlašovací údaje pak využije pro autentizaci vůči aktuálnímu poskytovateli členství. Tento poskytovatel bude nastaven pro autentizaci vůči uživatelským účtům v SQL databázi. Další aplikační událostí, která bude v modulu využita je PostAuthenticateRequest, která nastane po autentizaci daného požadavku. Pomocí ní dojde k autorizaci daného požadavku za využití http autorizačního modulu. Následně po úspěšné autentizaci a autorizaci dojde ke zpřístupnění session, do které se za pomoci aplikační události AcquireRequestState uloží potřebná data. Poslední aplikační událostí, která bude v modulu využita je EndRequest, která bude reagovat na případné stavové kódy http protokolu.



Obr. 6.1.1: Průběh komunikace

Jak je vidět na obr. 6.1.1, navržená komunikace klienta se serverem je následující:

1) Uživatel se pokouší přistoupit na privátní stránku Default.aspx, která je uložena v adresáři ~/Admin. Tento adresář má nastavená práva tak, že umožní přístup pouze uživatelům s rolí Admin. Jelikož je uživatel neautentizován, je mu přístup zamítnut a je přesměrován na přihlašovací stránku Login.aspx.

2) Uživatel zadá přihlašovací údaje, v události AuthenticateRequest HTTP modulu jsou tyto informace přečteny a zpracovány. Uživatel je ověřen proti nakonfigurovanému poskytovateli členství a pakliže je ověření úspěšné, vrací hodnotu True a uživatel je uložen do aktuálního kontextu *Application.Context.User*. Dále je zavolána funkce setAuthCookie a dojde tak k vygenerování autentizační cookie, která obsahuje náhodně vygenerované znaky pomocí funkce *RNGCryptoServiceProvider* a ty jsou spolu uživatelským jménem uloženy do databáze. Uživatel je přesměrován na privátní stránku.

3) Uživatel spolu s autentizační cookie požaduje privátní stránku Default.aspx. Komunikace je chráněna pomocí SSL, aby nebylo možné získat autnetizační cookie. v události AuthenticateRequest je zavolána funkce *getAuthCookie*, ze které se přečte dříve vygenerovaný řetězec znaků, dále se přečte z databáze příslušné uživatelské jméno to je uloženo do kontextu. V události PostAuthenticateRequest je přihlášený uživatel Autorizován. Pakliže je autorizace úspěšná, je povolen přístup na privátní stránku.

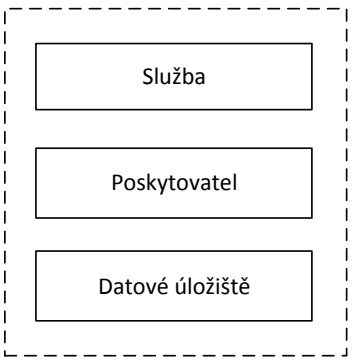
Proces autentizace je častým cílem útoků za účelem přístupu k citlivým datům. Tyto útoky jsou většinou směřovány na session ID, které slouží jako identifikátor uživatele na webu. Aplikace se proti útokům na session ID brání několika způsoby:

- Předávání session ID pomocí cookies s omezenou dobou platnosti
- Použití SSL
- Kontrola IP adresy a User-Agenta

Aby byla webová aplikace zabezpečena, je přenos šifrován pomocí SSL, to nabízí ochranu proti mnoha útokům, které spočívají ve sledování nezabezpečené komunikace. V takovém případě by bylo možné autnetizační cookie snadno odchytit a zneužít.
Dalším rizikovým místem je počítač vzdáleného uživatele, který může být kompromitován útočníkem, či nějakým škodlivým kódem. Pro tento případ autentizační modul odesílá cookie s příznakem HTTP-Only, což znemožňuje skriptům na klientově počítači k přístupu k cookie. Další formou zabezpečení je kontrola IP adresy a User-Agenta. Aby byl čas na provedení případného útoku omezen, je platnost cookie pevně nastavena na 20 minut.

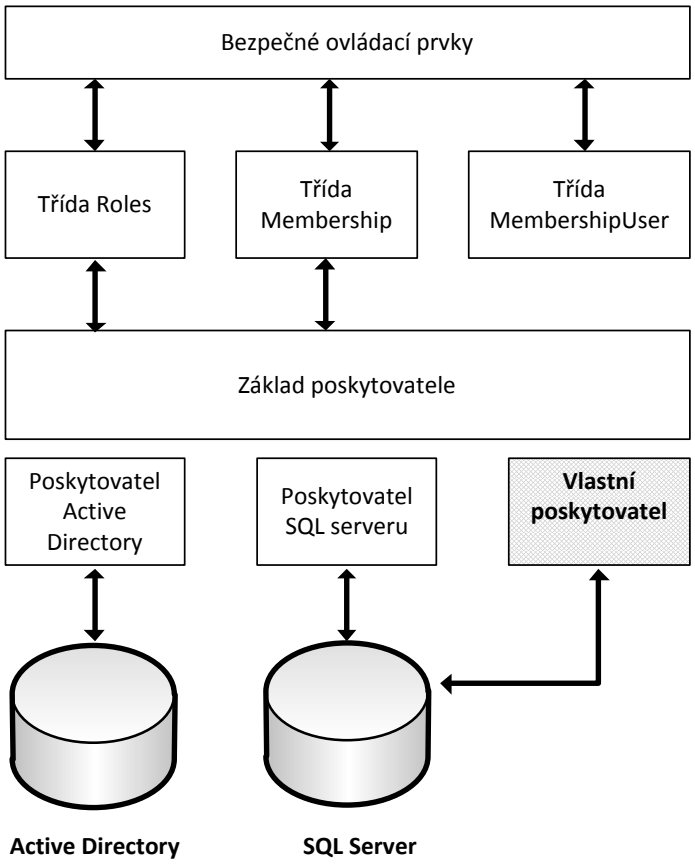
6.1.2 Vlastní poskytovatelé členství a rolí

Poskytovatelé jsou softwarové moduly, které tvoří mezistupeň mezi určitou službou a datovým úložištěm, které tato služba využívá. Touto službou mohou být profily, členství, role atd., které potřebují přístup k datovému úložišti. Z pohledu vícevrstvé architektury mohou být tyto dvě entity na sobě nezávislé, existuje-li mezistupeň – poskytovatel.



Obr. 6.1.2: Vícevrstvá architektura

ASP.NET obsahuje několik zabudovaných poskytovatelů, kteří jsou přizpůsobeni pro univerzální použití. Daní za tuto univerzálnost je vysoká složitost, proto v naší aplikaci byli vytvořeni vlastní poskytovatelé členství a rolí, kteří budou ukládat do databáze námi definované informace. Jako datové úložiště byl zvolen SQL Server, který je v Express edici obsažen ve vývojovém prostředí Visual Studia a výborně spolupracuje s ASP.NET.



Obr. 6.1.3: Poskytovatelé členství a rolí

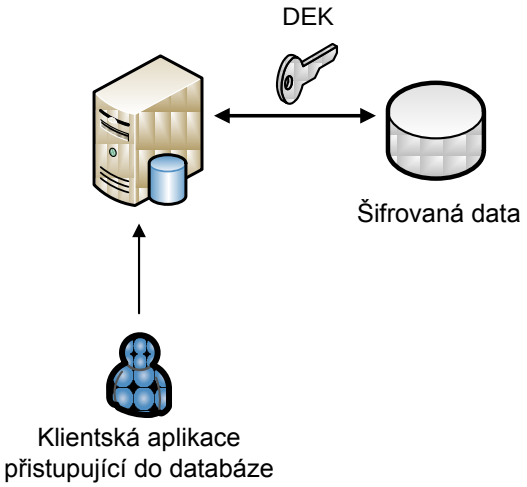
Jelikož poskytovatelé pracují s databází, je nutné zamezit útokům typu SQL Injection, toho je dosaženo využitím parametrizovaných SQL dotazů, které jsou automaticky ošetřeny a zbaveny nebezpečných znaků. Problematice těchto útoků se věnuje 5. kapitola.

Uživatelská data uložená v databázi mohou mít mít různé požadavky na bezpečnost. Pakliže chceme tato data uložená v databázi chránit, máme tři možnosti:

- Šifrování celé databáze
- Jednocestné zabezpečení obsahu
- Vratné metody zabezpečení obsahu

Šifrování celé databáze je nejbezpečnějším způsobem, jak ochránit citlivá data. V Enterprise edici SQL Serveru 2008 máme možnost využití transparentního šifrování, kdy je šifrován

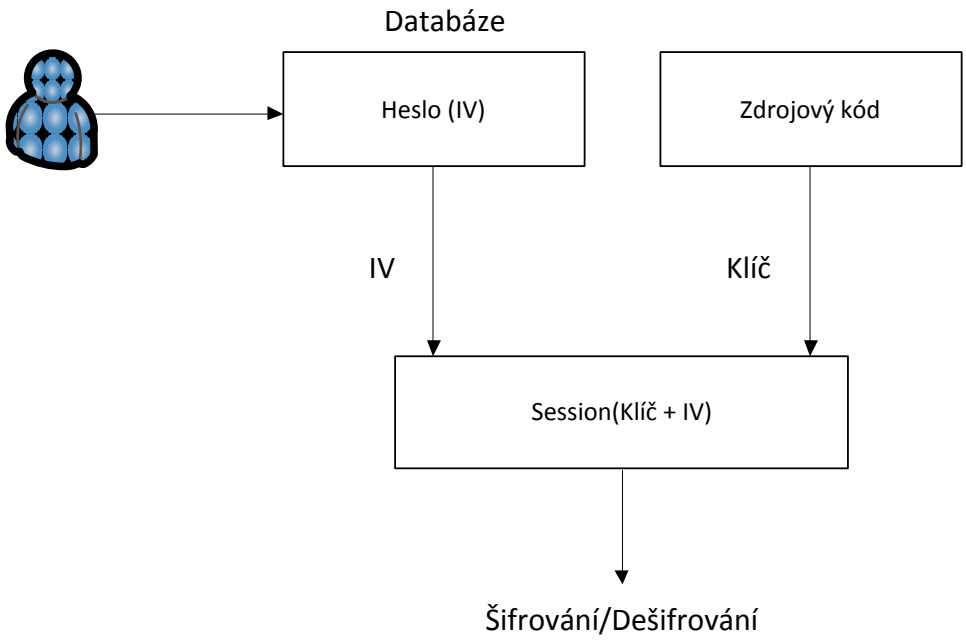
kromě databáze také transakční log, což má za následek to, že databázi není možné odpojit a zkopírovat. Pro zabezpečení databáze tímto způsobem je nutné nejprve vytvořit Master key, který je jedinečný pro celou databázi, následně vytvořit certifikát, který bude chránit šifrovací klíč databáze (DEK), pro který použijeme silný šifrovacích algoritmus AES256. Jelikož jsou v databázi uloženy citlivá data tisíců uživatelů, je možné pro zvýšení bezpečnosti použít hardwarové kryptografické moduly pro uložení klíčů.



Obr. 6.1.4: Princip transparentního šifrování dat

Jednocestné zabezpečení jsou nejčastěji využívána pro uložení hesla. Výhodou toho řešení je skutečnost, že nikdy není pracováno s textovou podobou hesla. Při registraci je do databáze uložen haš hesla a ten je následně během procesu přihlášení znovu spočítán a porovnán s uloženou hodnotou. Aby se zabránilo slovníkovým útokům, jsou tyt haše osoleny, to znamená, že je k nim přidán náhodně vygenerovaný řetězec.

Vratné metody zabezpečení jsou naopak využívány v případě, chceme-li získat šifrovanou hodnotu zpět v původním tvaru. K šifrování dat byla s ohledem na výkon zvolena symetrická bloková šifra AES. Data jsou šifrována za pomoci klíče, který bude uložen ve zdrojovém kódu, je tedy nutné jej chránit a inicializačního vektoru IV, který bude pro každého uživatele uložen v databázi a šifrován jeho heslem. Heslo uživatele je v databázi uloženo ve formě haše a tak jej není možné získat. Pakliže se uživatel přihlásí, je IV dešifrován za pomocí uživatelského hesla a spolu s klíčem uložen do session. Pomocí něj následně uživatel může provádět kryptografické operace (např. nahrát na server data a ta šifrovat apod.)



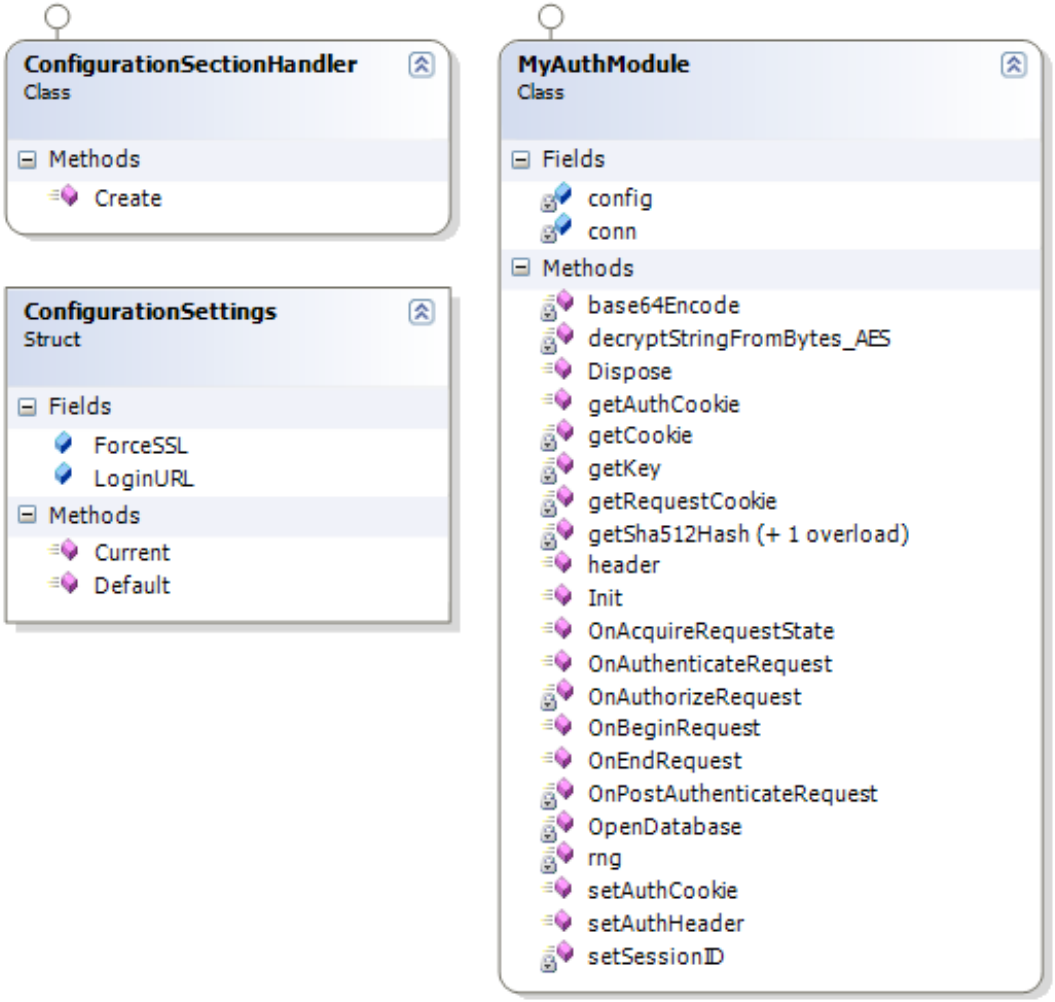
Obr. 6.1.5: Uložení klíče

6.2 Realizace navržené metody

Následující podkapitola se bude věnovat realizaci navržené autentizační metody za pomoci modulů a poskytovatelů.

6.2.1 Realizace autentizačního modulu

Autentizační modul je tvořen třídou MyAuthModule, která implementuje rozhraní *System.Web.IHttpModule*. Aby bylo možné modul konfigurovat, byly vytvořeny třídy ConfigurationSectionHandler a ConfigurationSettings. Strukturu těchto tříd zobrazuje obr. 6.2.1. Nyní budou blíže rozebrány jednotlivé třídy.



Obr. 6.2.1: Zvolená struktura stříd

MyAuthModule

Jedná se o hlavní třídu autentizačního modulu. Umožňuje zpracování jednotlivých aplikačních událostí, které budou podrobně rozebrány níže.

OnBeginRequest

reaguje na aplikační události BeginRequest. Jak již bylo zmíněno, umožňuje zapouzdření přihlašovacích údajů do autentizační hlavičky. Zde se nachází slabina dříve navržené metody, protože k potřebným událostem dochází až na straně serveru.Dále umožňuje přesměrování na bezpečné SSL spojení, jak zobrazuje níže uvedený zdrojový kód.

```
if (this.config.ForceSSL == "\"true\"")
{
      if (!Application.Request.IsSecureConnection)
      {
            string serverName =
            HttpUtility.UrlEncode(Application.Request.ServerVariables
            ["SERVER_NAME"]);
            string filePath = Application.Request.FilePath;
            Application.Response.Redirect("https://" + serverName +
            filePath);
      }
}
```

První podmínka slouží k aktivaci dané funkce prostřednictvím konfiguračního souboru. Ve druhé podmínce je testováno bezpečné *spojení*. Pakliže je shledáno, že spojení není bezpečné (např. zadá-li klient adresu ve formátu http://) dojde k přesměrování na šifrované spojení.

OnAuthenticateRequest

reaguje na aplikační událost AuthenticateRequest. Tato aplikační událost dekóduje autentizační hlavičku a ověří uživatele pomocí funkce *Membership.ValidateUser*. Ověření uživatele znázorňuje níže uvedený kód.

```
if (Membership.ValidateUser(UserName, Password))
{
      Application.Context.User = new ´
      System.Security.Principal.GenericPrincipal(new
      System.Security.Principal.GenericIdentity(UserName), newstring[0]);
      getKey(UserName, Password, context);
      if (requestCookie == null)
      {
            setAuthCookie(UserName, Application);
            object x = context.Items["key"];
            context.Items.Add("redirect", "true");
      }
}
```

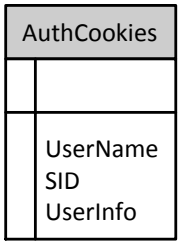
Pakliže uživatel přistupuje na danou webovou stránku poprvé, nepřísluší mu žádné cookies, proto je pomocí funkce *setAuthCookie* vytvořena autentizační cookie, která obsahuje vygenerované session ID, které slouží k identifikaci uživatele. Při každém požadavku na autentizaci následně probíhá test na přítomnost cookie za pomoci funkce *getAuthCookie*. Nyní budou podrobněji rozebrány využívané funkce.

setAuthCookie(string UserName, HttpApplication Application)

je funkce, která slouží k vytvoření autentizační cookie. Tato cookie je pojmenována jako ticket a obsahuje platné session ID. Z bezpečnostních důvodů má dobu platnosti 20 minut a nastavený parametr HttpOnly. Dále dojde ke zjištění IP adresy a User-Agenta klienta. Na tyto informace je následně aplikována hašovací funkce a jsou spolu se session ID a uživatelským jménem uloženy do databáze (viz obr. 6.2.2).

```
publicvoid setAuthCookie(string UserName, HttpApplication Application)
```

```
{
      int status;
      HttpCookie cookie = newHttpCookie("ticket");
      cookie.Expires = DateTime.Now.AddMinutes(1);
      cookie.Value = setSessionID(Application);
      cookie.HttpOnly = true;
      string kuki = cookie.Value;
      Application.Response.Cookies.Add(cookie);

      try
      {
            using (HostingEnvironment.Impersonate())
            {
                  using (SqlConnection conn = this.OpenDatabase())
                  using (SqlCommand cmd = new SqlCommand("IF NOT EXISTS
                  (SELECT * FROM AuthCookies WHERE UserName=@UserName)
                  INSERT INTO AuthCookies (UserName, SID, UserInfo)
                  VALUES (@UserName, @SID, @UserInfo) ELSE UPDATE
                  AuthCookies SET SID=@SID WHERE
                  UserName=@UserName", conn))
                  {
                        cmd.Parameters.Add("@UserName", SqlDbType.VarChar,
                        50).Value = UserName;
                        cmd.Parameters.Add("@SID",
                        SqlDbType.VarChar, 128).Value = kuki;

                        string ipAddress =
                        HttpContext.Current.Request.UserHostAddress;
                        string userAgent =
                        HttpContext.Current.Request.UserAgent;
                       string userInfo = getSha512Hash(ipAddress,
                        userAgent);

                        cmd.Parameters.Add("@UserInfo", SqlDbType.VarChar,
                        15).Value = userInfo;

                        int rowCount = cmd.ExecuteNonQuery();
                        if (rowCount == 0) status = 0;
                        else status = 1;
                  }
      }
}
catch { throw; }
}
```



Obr. 6.2.2: Tabulka pro uložení session ID

getAuthCookie(HttpApplication Application)

je funkce, která umožňuje přijmout autentizační cookie a přečíst z ní hodnotu session ID. Tato hodnota je následně použita k vyhledání daného uživatele spolu se zabezpečovacími informacemi v databázi. Následně je vypočítán haš z aktuální IP adresy a User-Agenta a ten je porovnám s hodnotou z databáze. Pakliže dojde ke shodě, je uživatelské jméno uloženo do kontextu a uživatel je přihlášen.

setSessionID(HttpApplication Application)

je funkce, která slouží ke generování session ID, které bude sloužit k identifikaci uživatele.. Aby byl tento vygenerovaný identifikátor uživatele skutečně náhodný, je použita kombinace aktuálního času, IP adresy klienta, verzí jeho prohlížeče a náhodně vygenerovaný řetězec. Na tyto hodnoty je následně aplikována hašovací funkce SHA-2, která nám vrátí 512 bitovou hodnotu. Tato hodnota je dostatečně dlouhá a náhodná na to, aby útočník v nebyl schopen vygenerovat kolizní platné session ID. Funkce, která má na starosti vygenerování náhodného session ID je uvedena níže.

```
privatestaticstring setSessionID(HttpApplication Application)
{
	string time = DateTime.Now.ToString("HH:mm:ss tt");
	string address = HttpContext.Current.Request.UserHostAddress;
	string browser = Application.Request.Browser.Version.ToString();

	Encoding enc = Encoding.Default;
	string random = enc.GetString(rng());
	string salt = base64Encode(random.ToString());

	string x;
	x = getSha512Hash(time, address, browser, salt);

	return x.ToString();
}
```

getKey(string UserName, string pwd, HttpContext context)

je funkce, která umožňuje získat inicializační vektor (IV), který je uložen v databázi a šifrován uživatelským heslem. Po přihlášení uživatele je s pomocí znalosti hesla dešifrován a uložen do aktuálního kontextu.

OnPostAuthenticateRequest

reaguje na aplikační událost PostAuthenticateRequest a za pomoci funkce *UrlAuthorizationModule.CheckUrlAccessForPrincipal* umožňuje autentizovanému uživateli přístup k požadované stránce dle nastavených přístupových práv. Tato práva jsou nastavena v sekci *authorization* konfiguračního souboru web.config a mají následující formát:

```
<authorization>
	<denyusers="?" />
	<allowusers="*" />
```

```
      <allowroles="User" />
      <denyusers="*" />
</authorization>
```

OnAcquireRequestState

reaguje na aplikační událost AcquireRequestState a umožňuje přístup k session. V této události jsou uložena uživatelská data do session, jak je patrné z níže uvedeného kódu.

```
if (Application.Context.User.Identity.IsAuthenticated == true)
{
      Application.Context.Session["isAuthenticated"] = "true";
      Application.Context.Session["UserName"] =
      Application.Context.User.Identity.Name;
      Application.Context.Session["Role"] =
      Roles.GetRolesForUser(Application.Context.User.Identity.Name);
      if (context.Items["redirect"] == "true")
      {
            context.Session["test"] = context.Items.Contains("key");
            context.Session["test"] = context.Items["key"];
            Application.Response.Redirect("~/Admin/Default.aspx");
      }
}
```

OnEndRequest

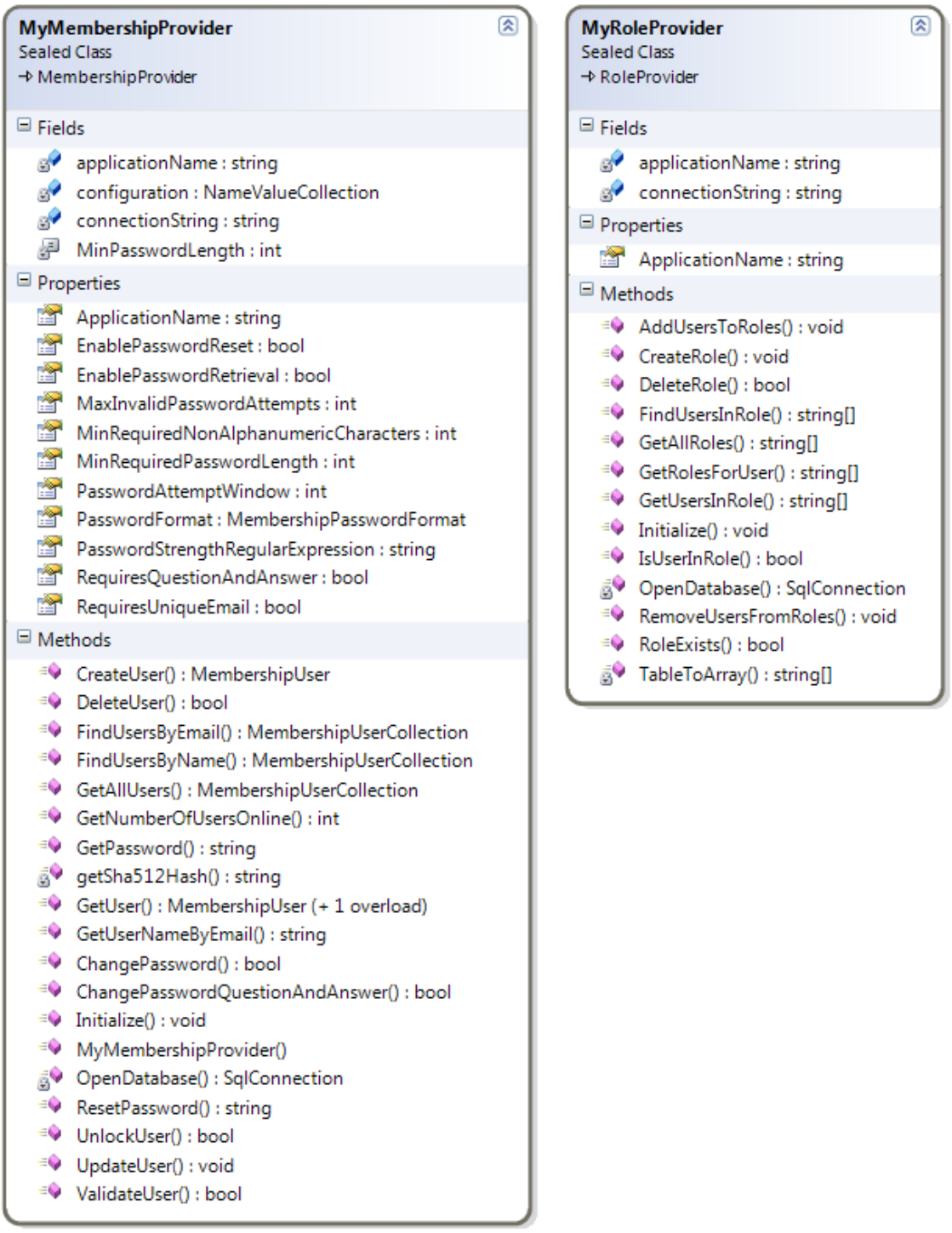
Reaguje na aplikační událost EndRequest a slouží ke zpracování stavových kódů http protokolu. Pakliže je zaznamenán kód 401, dojde pomocí funkce *httpapplication.Response.Redirect* k přesměrování na definovanou přihlašovací stránku.

ConfigurationSectionHandler je obslužná rutina, která umožňuje přidat do souboru web.config vlastní konfogurační sekci, v našem případě se jedná o nakonfigurování přihlašovací stránky.

ConfigurationSettings je struktura, která slouží k uchování konfiguračních informací.

6.2.2 Realizace poskytovatelů

Poskytovatelé jsou tvořeni dvěmi třídami, MyMembershipProvider, která implementuje rozhraní MembershipProvider a MyRoleProvider, která implementuje rozhraní MyRoleProvider. Strukturu jednotlivých tříd zobrazuje obr. 6.2.3.Nyní budou blíže rozebrány obě třídy.



Obr. 6.2.3: Třídy pro poskytovatele.

MyMembershipProvider

Jedná se o třídu, která má na starosti členství. Obsahuje několik metod, které umožňují správu uživatelských účtů, které jsou uloženy v databázi. Nyní budou blíže rozebrány metody, které byly v práci implementovány.

CreateUser

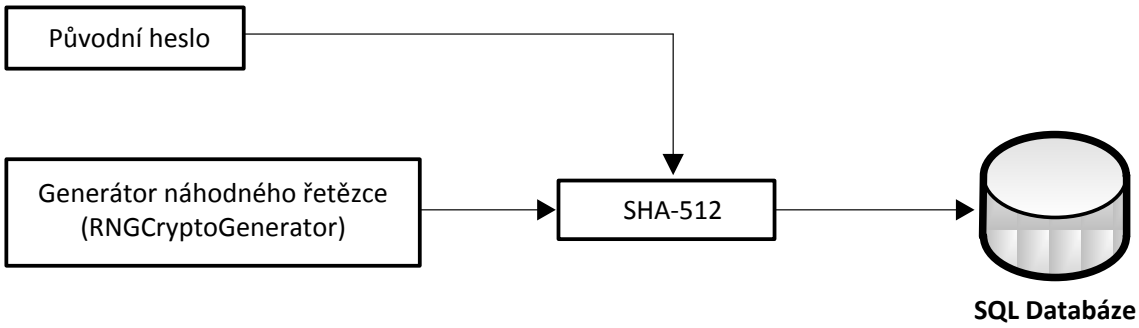
je metoda, která umožňuje vytvořit uživatelský účet a uložit jej do databáze. Před uložením je heslo zkontrolováno, zda-li splňuje minimální požadavky na bezpečnost, tedy minimální délku a počet speciálních znaků.

Do databáze se ukládají následující hodnoty:

- Uživatelské jméno
- Heslo v podobě haše
- Sůl
- E-mail
- Inicializační vektor šifrovaný pomocí uživatelského hesla

Jak již bylo zmíněno, heslo se do databáze ukládá v hašované podobně (viz lit. [4]). K tomu je využito hašovací funkce SHA-2. Jako sůl byl zvolen náhodně vygenerovaný řetězec znaků pomocí funkce *RNGCryptoServiceProvider*.

```
Private static string getSha512Hash(string input, string salt)
{
      SHA512Managed Sha512Hasher = newSHA512Managed();
      string pwdsalt = String.Concat(input, salt);
      byte[] data =
      Sha512Hasher.ComputeHash(Encoding.Default.GetBytes(pwdsalt));
      StringBuilder sBuilder = newStringBuilder();
      for (int i = 0; i < data.Length; i++)
      {
            sBuilder.Append(data[i].ToString("x2"));
      }
      return sBuilder.ToString();
}
```



Obr. 6.2.4: Způsob uložení osoleného hašovaného hesla do databáze

ValidateUser

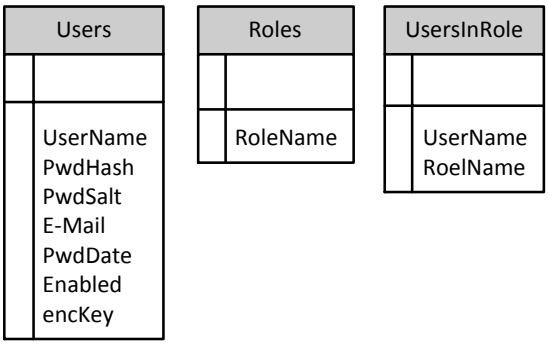
Jedná se o metodu, která slouží k ověření uživatele vůči databázi. To je provedeno tak, že se z aktuálně zadaného hesla spočítá haš a ten je porovnán s hodnotou uloženou v databázi. Pokud jsou hodnoty totožné, byl uživatel úspěšně autentizován.

```
Public override bool ValidateUser(string username, string password)
{
      if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))
      return false;

      username = username.ToLower();

      string PwdHash;
      string PwdSalt;

      try
      {
            using (HostingEnvironment.Impersonate())
            using (SqlConnection conn = this.OpenDatabase())
            using (SqlCommand cmd = new SqlCommand("SELECT PwdHash, PwdSalt
            FROM Users WHERE UserName=@UserName AND Enabled=1", conn))
            {
                  cmd.Parameters.Add("@UserName", SqlDbType.VarChar,
                  50).Value = username;

                  using (SqlDataReader reader =
                  cmd.ExecuteReader(CommandBehavior.SingleRow))
                  {
                        if (!reader.Read()) returnfalse;

                        PwdHash = reader["PwdHash"] asstring;
                        PwdSalt = reader["PwdSalt"] asstring;
                  }
            }
      }
      catch { throw; }

      if (getSha512Hash(password, PwdSalt).Equals(PwdHash,
      StringComparison.Ordinal))
      {
            return true;
      }
      else return false;
}
```

MyRoleProvider

Jedná se o třídu, která má na starosti správu rolí. umožňuje přidat a odebrat uživatele z rolí, k tomu slouží metody *GetUsersInRoles* a *RemoveUsersFromRoles*. K ověření, zda-li je uživatel v dané roli slouží metoda *GetRolesForUser*.

Pro uložení daných informací byla vytvořena následující struktura tabulek (viz obr. 6.2.5). V tabulce Users jsou uloženy informace potřebné pro autentizaci uživatele. Následující tabulky Roles a UserInRole souvisejí s autorizací uživatele.



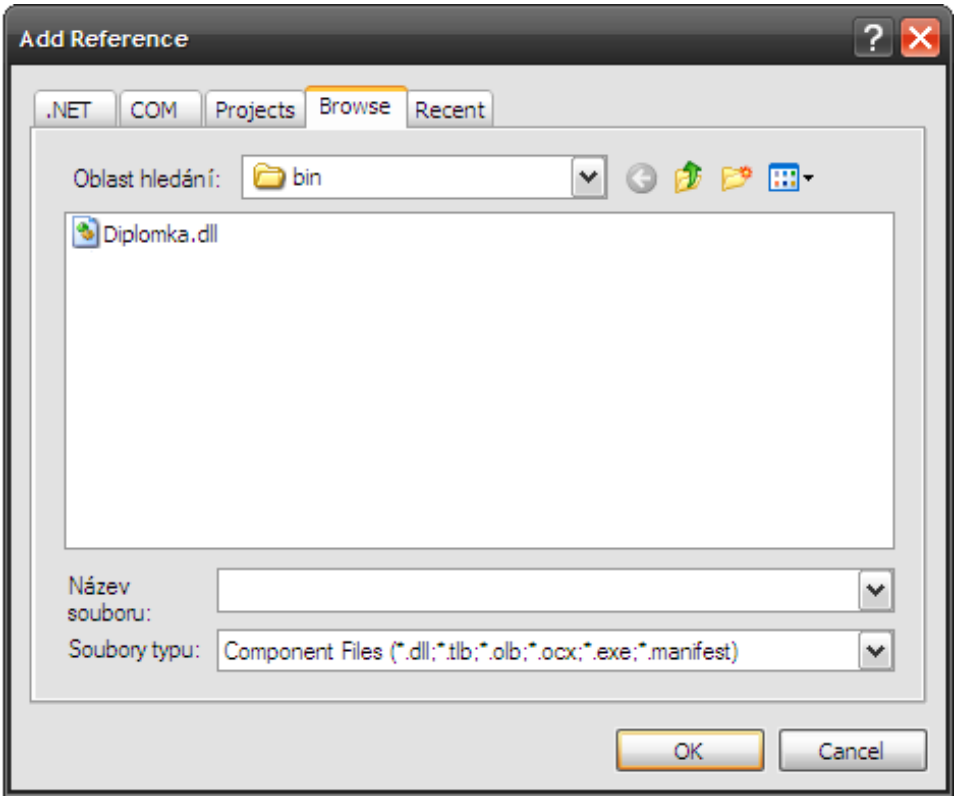
Obr. 6.2.5: Struktura tabulek v databázi

6.3 Zprovoznění HTTP modulu

Aby bylo možné zprovoznit modul, je nutné udělat dvě věci:

- Přidat referenci na soubor Diplomka.dll
- Přidat modul a příslušné poskytovatele do konfiguračního souboru web.config

Přidání reference na soubor provedeme ve Visual Studiu tak, že v hlavním menu Project zvolíme položku Add Reference a v dialogovém okně viz obr. 6.3.1 pak vybereme potřebný soubor. Následně by se v okně Solution Exploreru v položce References měl objevit objekt Diplomka.



Obr. 6.3.1: Přidání reference

Dále je nutné v konfiguračním souboru Web.config, který se nachází v kořenovém adresáři projektu, přidat modul a patřičné poskytovatele.

Přidání modulu provedeme přidáním níže uvedeného kódu do sekce <system.web>

```
<httpModules>
      <addname="MyAuthModule"type="Diplomka.MyAuthModule, Diplomka"/>
</httpModules>
```

Dále je nutné vypnout integrovanou autentizaci v ASP.NET

```
<authenticationmode="None"/>
```

Nyní přidáme poskytovatele členství, ten se stejně jako modul přidá do sekce <system.web> vložením následujícího kódu

```
<membershipdefaultProvider="MyMembershipProvider">
      <providers>
            <clear />
            <addname="MyMembershipProvider"
type="Diplomka.MyMembershipProvider, Diplomka"
connectionStringName="MyConnectionString" />
      </providers>
</membership>
```

A dále přidáme do stejné sekce poskytovatele rolí

```
<roleManagerenabled="true"defaultProvider="MyRoleProvider">
      <providers>
            <clear/>
            <addname="MyRoleProvider"
                  type="Diplomka.MyRoleProvider, Diplomka"
                  connectionStringName="MyConnectionString" />
      </providers>
</roleManager>
```

Tímto je konfigurace modulu a příslušných poskytovatelů hotová a zbývá už jen vytvořit databáze a příslušné tabulky.

6.4 Zachycení komunikace

Pro znázornění průběhu komunikace byl použit nástroj Fiddler (viz lit [20]), který je možné volně stáhnout z internetu fiddlertool.com. Tento nástroj umožňuje zachycení http komunikace prostřednictvím integrovaného proxy serveru.

Nyní budou zobrazeny výsledky monitorování provozu. Aby byly výsledky čitelné, nebyl síťový provoz šifrován pomocí SSL.

Odeslání přihlašovacích údajů

Nejprve jsou odeslány na server přihlašovací údaje z formuláře. Ze zachycené komunikace je patrné, že jsou zakódované ve formátu base 64.

```
POST /Secure/Login.aspx HTTP/1.1
Host: localhost:50586
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; cs; rv:1.9.1.9)
Gecko/20100315 Firefox/3.5.9
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: cs,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: windows-1250,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://localhost:50586/Secure/Login.aspx
Content-Type: application/x-www-form-urlencoded
Content-Length: 1907

__EVENTTARGET=&__EVENTARGUMENT=&__VIEWSTATE=%2FwEPDwUKMTcxMzMwNzcxMWQYAwUeX
19Db250cm9sc1JlcXVpcmVQb3N0QmFja0tleV9fFg0FEUxvZ2luMSRSZW1lbWJlck1lBRdMb2dp
..
..
mRNdWx0aVZpZXcPD2RmZN8LZzBuKRQoAm5fSH2YLBICqBOb&__EVENTVALIDATION=%2FwEWDAL
gjrXBDQKUvNa1DwL666vYDAKC0q%2BkBgKnz4ybCALfquvXCgK4%2BbCJAwKOsenbBwLL5MPXBQ
KT99a2DgLAjfbKCwKDpevXCkyLqObQjTX6GdfJBP0weyBacm%2Fv&Login1%24UserName=test
2&Login1%24Password=test.321&Login1%24LoginButton=P%C5%99ihl%C3%A1sit&Creat
eUserWizard1%24CreateUserStepContainer%24UserName=&CreateUserWizard1%24Crea
teUserStepContainer%24Password=&CreateUserWizard1%24CreateUserStepContainer
%24ConfirmPassword=&CreateUserWizard1%24CreateUserStepContainer%24Email=&Cr
eateUserWizard1%24CreateUserStepContainer%24Question=&CreateUserWizard1%24C
reateUserStepContainer%24Answer=
```

Odezva serveru

Po úspěšné autentizaci jsou klientovi zaslány cookie, autentizační s názvem ticket a cookie pro session. Dojde také k přesměrování na privátní stránku, to je indikování stavovým kódem HTTP/1.1 302..

```
HTTP/1.1 302 Found
Server: ASP.NET Development Server/9.0.0.0
Date: Sat, 22 May 2010 15:12:52 GMT
X-AspNet-Version: 2.0.50727
Location: /Admin/Default.aspx
Set-Cookie:
ticket=c0cfb154a1cdeb56a453343cdc400c3ce736be5198ccbeca125e81531c9727b9d681
3706d2a64d60ed91a098bb698311196307fcc26bb9be9d60142eb90e758d;  expires=Sat,
22-May-2010 15:13:52 GMT; path=/; HttpOnly
Set-Cookie: ASP.NET_SessionId=uimdlfyhnpktakrxa5toweed; path=/; HttpOnly
Cache-Control: private
```

```
Content-Type: text/html; charset=utf-8
Content-Length: 140
Connection: Close
```

Požadavek na zabezpečenou stránku

Klient provede požadavek na zabezpečenou stránku včetně cookie, které jsou uloženy v http záhlaví.

```
GET /Admin/Default.aspx HTTP/1.1
Host: localhost:50586
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; cs; rv:1.9.1.9)
Gecko/20100315 Firefox/3.5.9
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: cs,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: windows-1250,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://localhost:50586/Secure/Login.aspx
Cookie:
ticket=c0cfb154a1cdeb56a453343cdc400c3ce736be5198ccbeca125e81531c9727b9d681
3706d2a64d60ed91a098bb698311196307fcc26bb9be9d60142eb90e758d;
ASP.NET_SessionId=uimdlfyhnpktakrxa5toweed
```

Odezva serveru

Po úspěšné autentizaci klient přistoupil na privátní stránku. To je indikováno stavovým kódem HTTP/1.1 200.

```
HTTP/1.1 200 OK
Server: ASP.NET Development Server/9.0.0.0
Date: Sat, 22 May 2010 15:12:54 GMT
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1501
Connection: Close
```

ZÁVĚR

Tématem práce je návrh a realizace autentizační metody. Při jejím návrhu byly kladeny požadavky na bezpečnost a uživatelskou přívětivost v podobě přihlašovacího formuláře, který by bylo možné zakomponovat do stávajícího systému webových stránek. Daným požadavkům nejlépe vyhovuje možnost využití http autentizace spolu s formulářovým přihlašováním. Takto navržená metoda měla být realizována jako HTTP modul.

Nicméně tento cíl se nepodařilo splnit, jelikož všechny události, které modul obsluhuje, probíhají přímo na serveru a pro realizaci navřené metody jej nelze použít. Z toho důvodu byla realizována prostřednictvím HTTP modulu vlastní formulářová autentizace.

Tato autentizační metoda, na rozdíl od integrované formulářové autentizace ASP.NET umožnuje eliminovat útoky na session ID tím, že provádí kontrolu identity. Při generování session ID jsou uloženy informace o vlastníkovi a ty jsou během každého požadavku ověřovány. Liší-li se tyto informace, aplikace považuje přístup ke zdroji za pokus o útok a dojde tak k odepření přístupu. Tímto lze eliminovat případné útoky session

Další částí práce bylo zabezpečení uživatelských dat na serveru. To bylo realizováno za pomoci šifry AES. Klíč je zakomponován ve zdrojovém kódu a inicializační vektor je chráněn pomocí hesla uživatele. To je v databázi uloženo ve formě haše, a tak nelze tajnou informaci získat v případě jejího zneužití.

Nevýhodou realizované metody je nutnost šifrování komunikace za pomoci SSL, které má negativný dopad na výkon celého serveru, protože server musí udržovat velké množství šifrovaných spojení. Zde se nabízí možnost použití hardwareového SSL akcelerátoru, který by tento nedluh odstranil. Další nevýhoda zvolené metody spočívá v případném použití technologie NAT na straně klienta. Ta má za následek oslabení zabezpečení v případě útoku z vnitřní sítě.

LITERATURA

[1] EVJEN, B., HANSELMAN, S., RADER, D.: *Professional ASP.NET 3.5: In C# and VB*, Wrox 2008, ISBN: 978-0-470-18757-9

[2] MACDONALD, M., SZPUSZTA, M.: *Pro ASP.NET 3.5 in C# 2008*, Apress 2007, ISBN: 978-1-59059-893-1

[3] VOLODARSKY, M.: *Internet Information Services (IIS) 7.0 Resource Kit*, Microsoft Press 2008, ISBN: 978-0-73562-441-2

[4] MSDN MAGAZINE: *Hashing Passwords* [online]. 2003 [cit. 1.11.2009] Dostupný z WWW: <http://msdn.microsoft.com/en-us/magazine/cc164107.aspx>

[5] DOSEDĚL, T.: *Počítačová bezpečnost a ochrana dat*, Computer 2004, ISBN 80-251-0106-1

[6] MSDN LIBRARY: *ASP.NET Session State Overview* [online]. [cit. 1.11.2009] Dostupný z WWW <http://msdn.microsoft.com/en-us/library/ms178581.aspx>

[7] MSDN LIBRARY: *ASP.NET State Management Recommendations* [online]. [cit. 1.11.2009] Dostupný z WWW <http://msdn.microsoft.com/en- us/library/ z1hkazw7.aspx>

[8] THE CODE PROJECT: *Exploring Session in ASP.NET* [online]. [cit. 1.11.2009] Dostupný z WWW < http://www.codeproject.com/KB/aspnet/ExploringSession.aspx>

[9] LACKO, L.: *ASP.NET a ADO.NET 2.0 Hotová řešení*, Computer Press 2006, ISBN: 80-251-1028-1

[10] KOLŠEK, M.: *Session Fixation Vulnerability in Web-based Applications* [online]. 2007 [cit. 1.11.2009] Dostupný z WWW <http://www.acrossecurity.com/ papers/ session_fixation.pdf>

[11] MICROSOFT CONNECT: *Session fixation* [online]. [cit. 1.11.2009] Dostupný z WWW < https://connect.microsoft.com/VisualStudio/feedback/ViewFeedback.aspx? FeedbackID=143361 >

[12] KNIGHT, B.: *Professional Microsoft SQL Server 2008 Administration*, Wrox 2008, ISBN: 978-0-470-24796-9

[13] MICROSOFT: *SQL Server 2008* [online]. [cit. 1.11.2009] Dostupný z WWW <http://www.microsoft.com/cze/sqlserver2008/security.mspx>

[14] MSDN LIBRARY: *ASP.NET Session State Modes* [online]. [cit. 1.11.2009] Dostupný z WWW <http://msdn.microsoft.com/en-us/library/ms178586.aspx>

[15] KRAUSE, J.: *Pro ASP.NET Extensibility*, Wrox 2009, ISBN: 978-1-4302-1984-2

[16] VALASEK, M.: Modul pro basic autentizaci v ASP.NET [online]. [cit. 1.3.2010] Dostupný z WWW <http://www.aspnet.cz/Articles/84-modul-pro-basic-autentizaci-v-asp-net.aspx>

[17] LEVICKÝ, D.: *Kryptografia v informačnej bezpečnosti*, Elfa 2005, ISBN:80-8086-022-X

[18] MSDN LIBRARY: *SQL Injection* [online]. [cit. 17.4.2010] Dostupný z WWW <http://msdn.microsoft.com/en-us/library/ms161953.aspx>

[19] MSDN LIBRARY: *Forms Authentication* [online]. [cit. 17.4.2010] Dostupný z WWW <http://msdn.microsoft.com/en-us/library/system.web.security .formsauthenticationticket.aspx>

[20] FIDDLER*: Web Debugging Proxy* [online]. [cit. 15.5.2010] Dostupný z WWW < http://www.fiddler2.com/fiddler2/ >